US006247057B1

(12) **United States Patent**
Barrera, III

(10) **Patent No.: US 6,247,057 B1**
(45) **Date of Patent: Jun. 12, 2001**

(54) **NETWORK SERVER SUPPORTING MULTIPLE INSTANCE OF SERVICES TO OPERATE CONCURRENTLY BY HAVING ENDPOINT MAPPING SUBSYSTEM FOR MAPPING VIRTUAL NETWORK NAMES TO VIRTUAL ENDPOINT IDS**

(75) Inventor: **Joseph S. Barrera, III**, San Bruno, CA (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/177,984**

(22) Filed: **Oct. 22, 1998**

(51) **Int. Cl.**[7] **G06F 15/16**
(52) **U.S. Cl.** **709/229**; 709/203
(58) **Field of Search** 709/229, 203

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,568,487 * 10/1996 Sitbon et al. 370/466
5,636,371 * 6/1997 Yu 709/227
5,734,865 * 3/1998 Yu 709/250
5,881,269 * 3/1999 Dobbelstein 395/500.42
6,041,166 * 3/2000 Hart et al. .

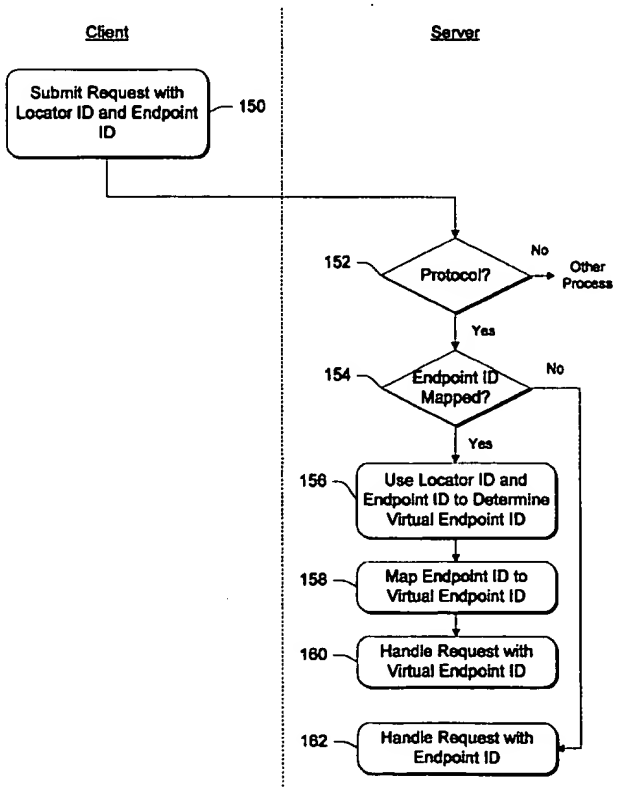
* cited by examiner

Primary Examiner—Krisna Lim
(74) *Attorney, Agent, or Firm*—Lee & Hayes PLLC

(57) **ABSTRACT**

In a network server that supports multiple instances of the same service, clients individually request the virtual services by using virtual network names assigned to the virtual services. The virtual network names include a locator ID (e.g., server name, IP address) and a designated endpoint ID (e.g., named pipe name, port ID). The server implements an endpoint mapping subsystem that creates new virtual endpoint IDs that are different from the designated endpoint IDs and associates the virtual endpoint IDs with the virtual network names. When a client request is received at the server, the endpoint mapping subsystem examines the virtual network name, looks up the corresponding virtual endpoint ID, and replaces the virtual network name with the virtual endpoint ID in the request. This allows more than one instance of the service to operate concurrently.

32 Claims, 7 Drawing Sheets



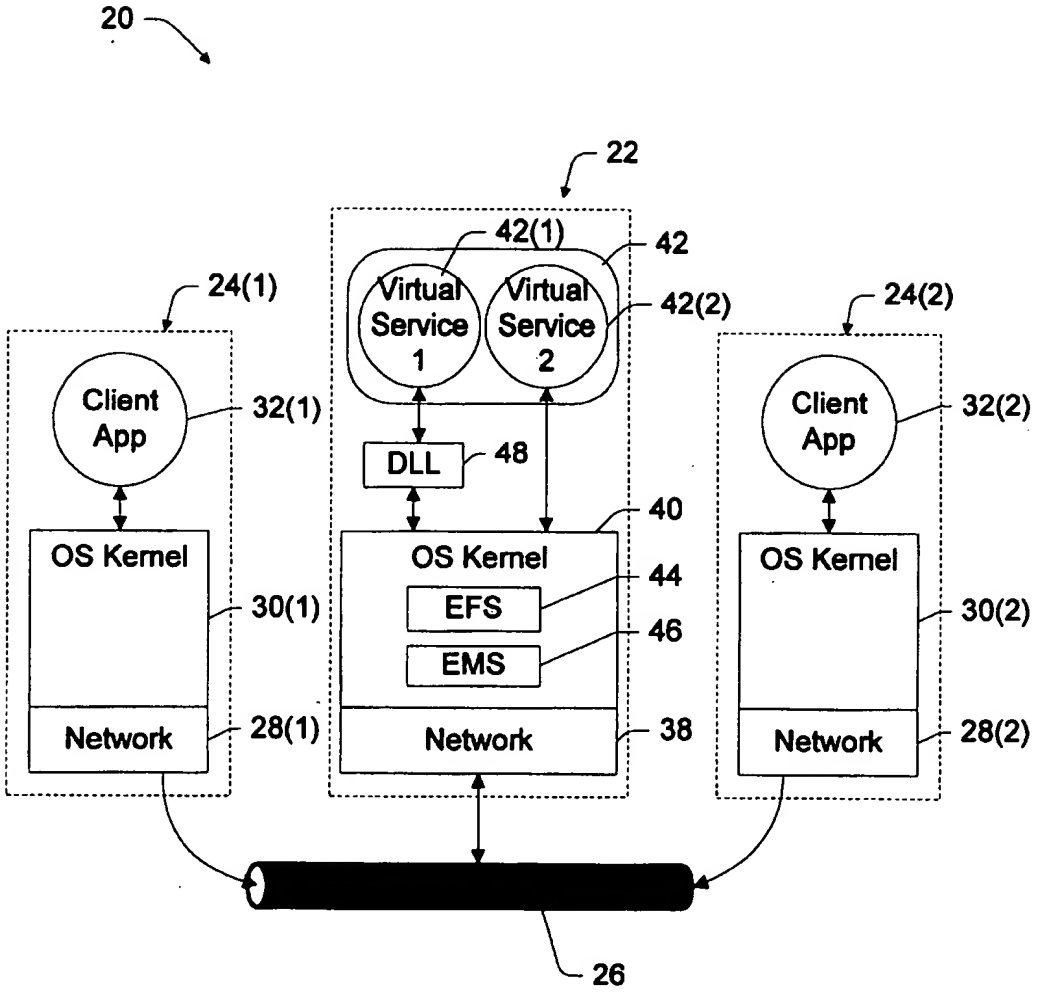

20
22
42(1)
42
Virtual Service 1
Virtual Service 2
42(2)
24(1)
24(2)
Client App
32(1)
Client App
32(2)
DLL
48
40
OS Kernel
OS Kernel
OS Kernel
44
EFS
46
EMS
30(1)
30(2)
Network
28(1)
Network
38
Network
28(2)
26

Fig. 1

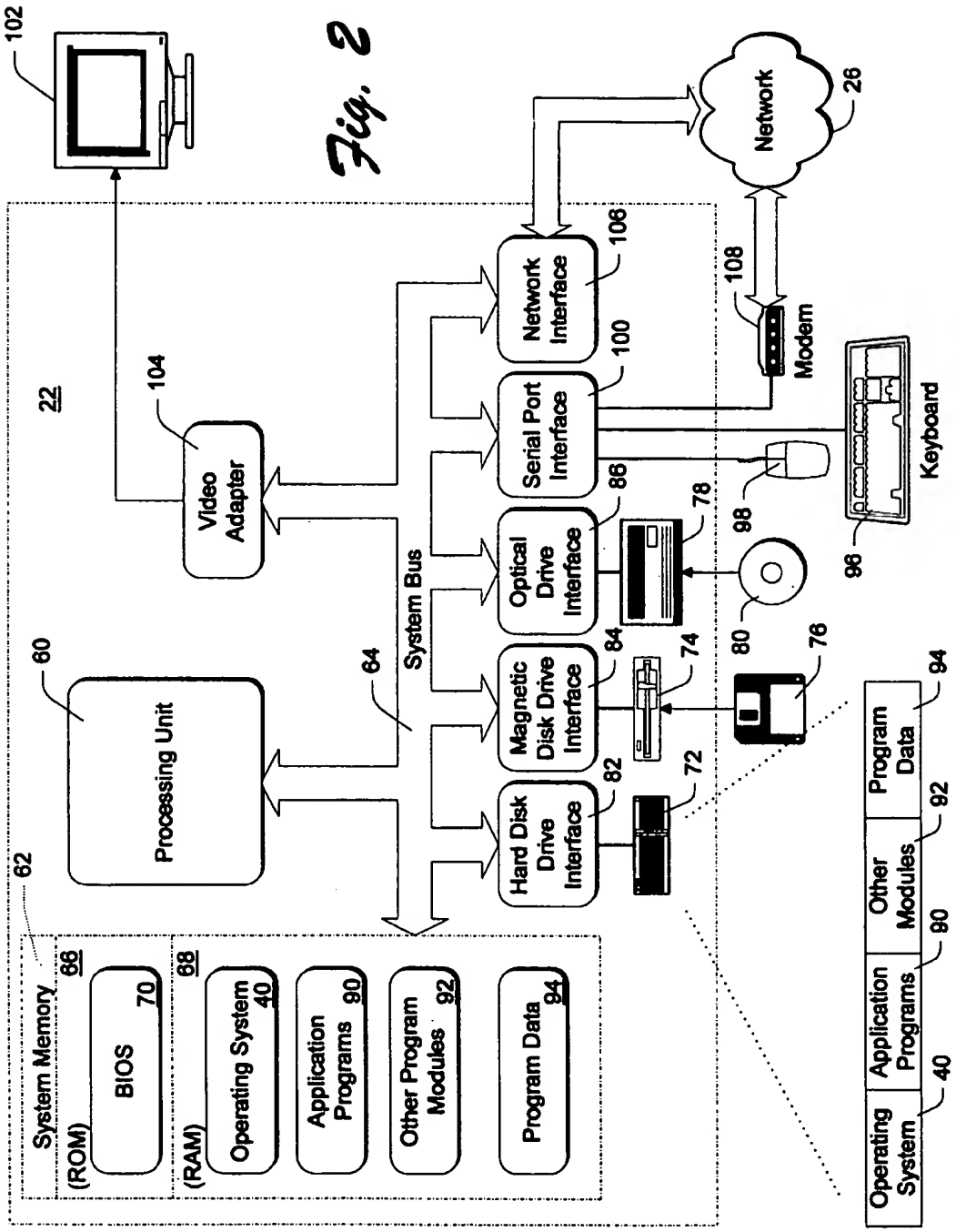


Fig. 2
102
Network
26
22
104
Video Adapter
Network Interface
106
108
Modem
Serial Port Interface
100
Keyboard
98
96
Optical Drive Interface
86
78
80
System Bus
64
Magnetic Disk Drive Interface
84
74
76
60
Processing Unit
Hard Disk Drive Interface
82
72
62
System Memory
66
(ROM)
BIOS
70
68
(RAM)
Operating System
40
Application Programs
90
Other Program Modules
92
Program Data
94
Program Data
94
Other Modules
92
Application Programs
90
Operating System
40

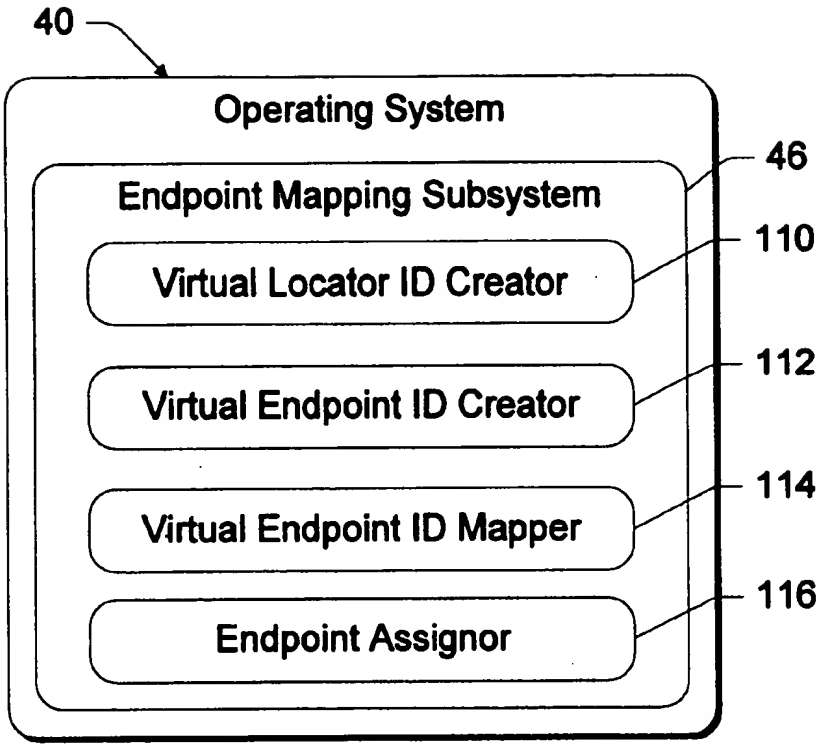

40
Operating System
46
Endpoint Mapping Subsystem
110
Virtual Locator ID Creator
112
Virtual Endpoint ID Creator
114
Virtual Endpoint ID Mapper
116
Endpoint Assignor

Fig. 3

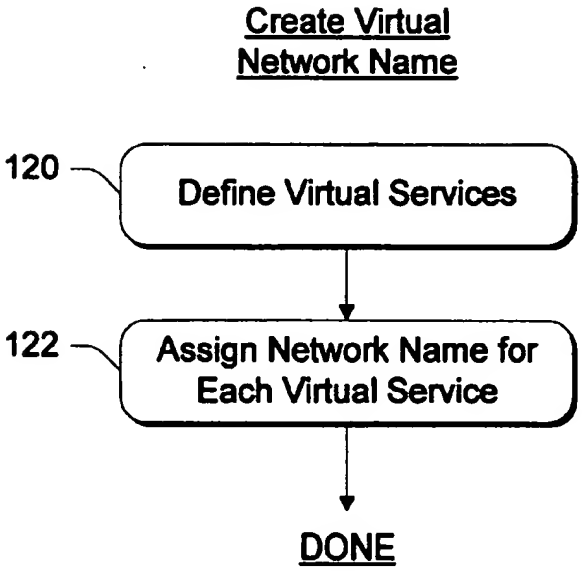

Create Virtual Network Name
120
Define Virtual Services
122
Assign Network Name for Each Virtual Service
DONE

Fig. 4

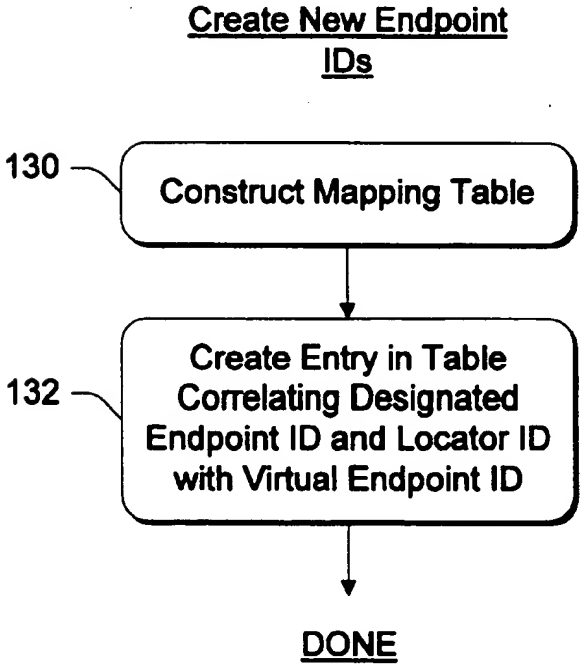

Create New Endpoint IDs
130
Construct Mapping Table
132
Create Entry in Table Correlating Designated Endpoint ID and Locator ID with Virtual Endpoint ID
DONE

Fig. 5

140

Locator ID	Designate Endpoint ID	Virtual Endpoint ID

Fig. 6a

142

Server Name	Named Pipe Name	Virtual Endpoint ID
virtual_server1	pipe\SQL\query	\pipe\$$\virtual_server1\SQL\query
virtual_server2	pipe\SQL\query	\pipe\$$\virtual_server2\SQL\query

Fig. 6b

144

IP Address	Port ID	Virtual Endpoint ID
129.56.85.7	80	129.56.85.7:81
129.56.85.8	80	129.56.85.8:82

Fig. 6c

146

Port ID	Mapped
80	Y
90	N

144

IP Address	Port ID	Virtual Endpoint ID
129.56.85.7	80	129.56.85.7:81
129.56.85.8	80	129.56.85.8:82

Fig. 6d

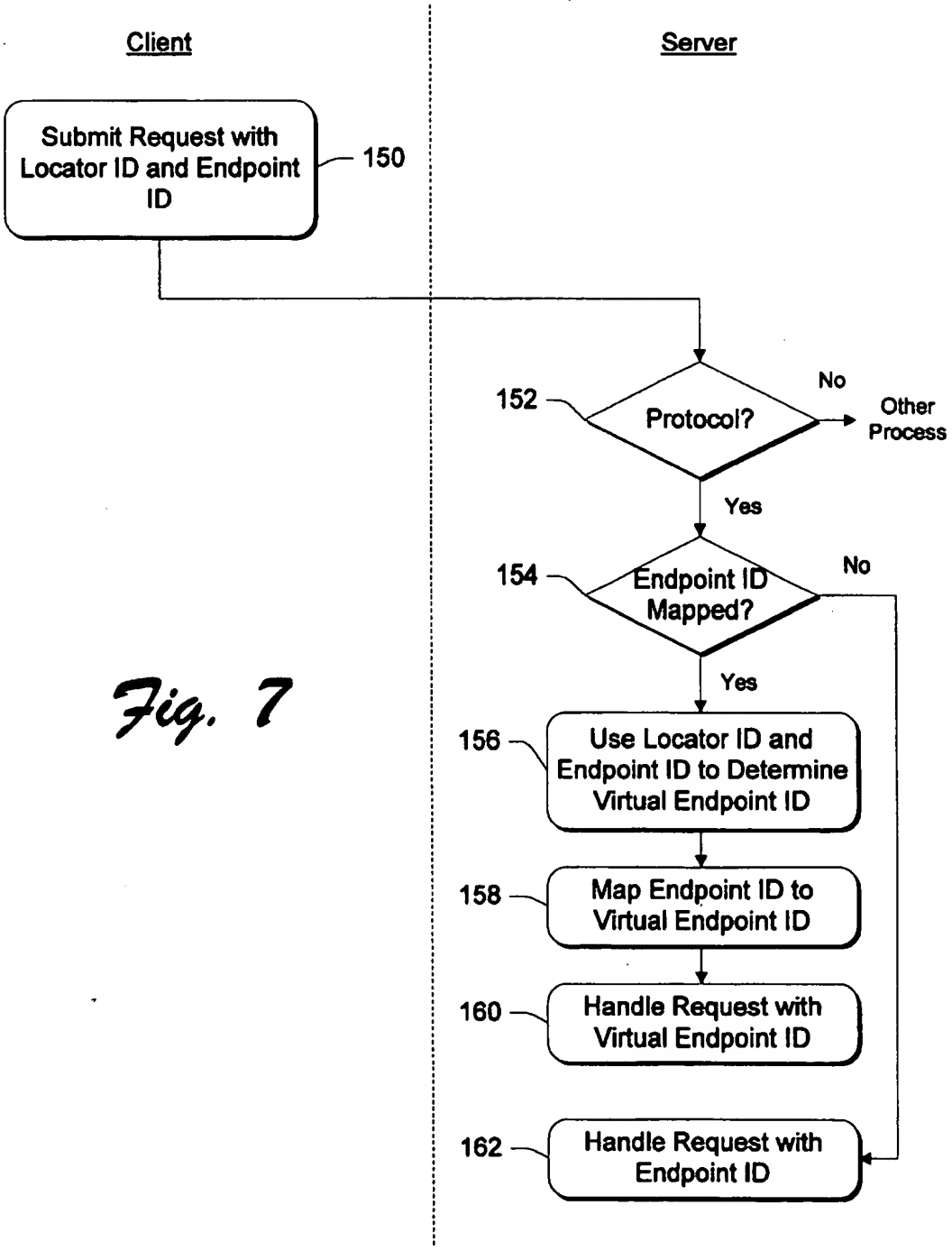

Client
Server
Submit Request with Locator ID and Endpoint ID
150
152
Protocol?
No
Other Process
Yes
154
Endpoint ID Mapped?
No
Yes
156
Use Locator ID and Endpoint ID to Determine Virtual Endpoint ID
158
Map Endpoint ID to Virtual Endpoint ID
160
Handle Request with Virtual Endpoint ID
162
Handle Request with Endpoint ID

Fig. 7

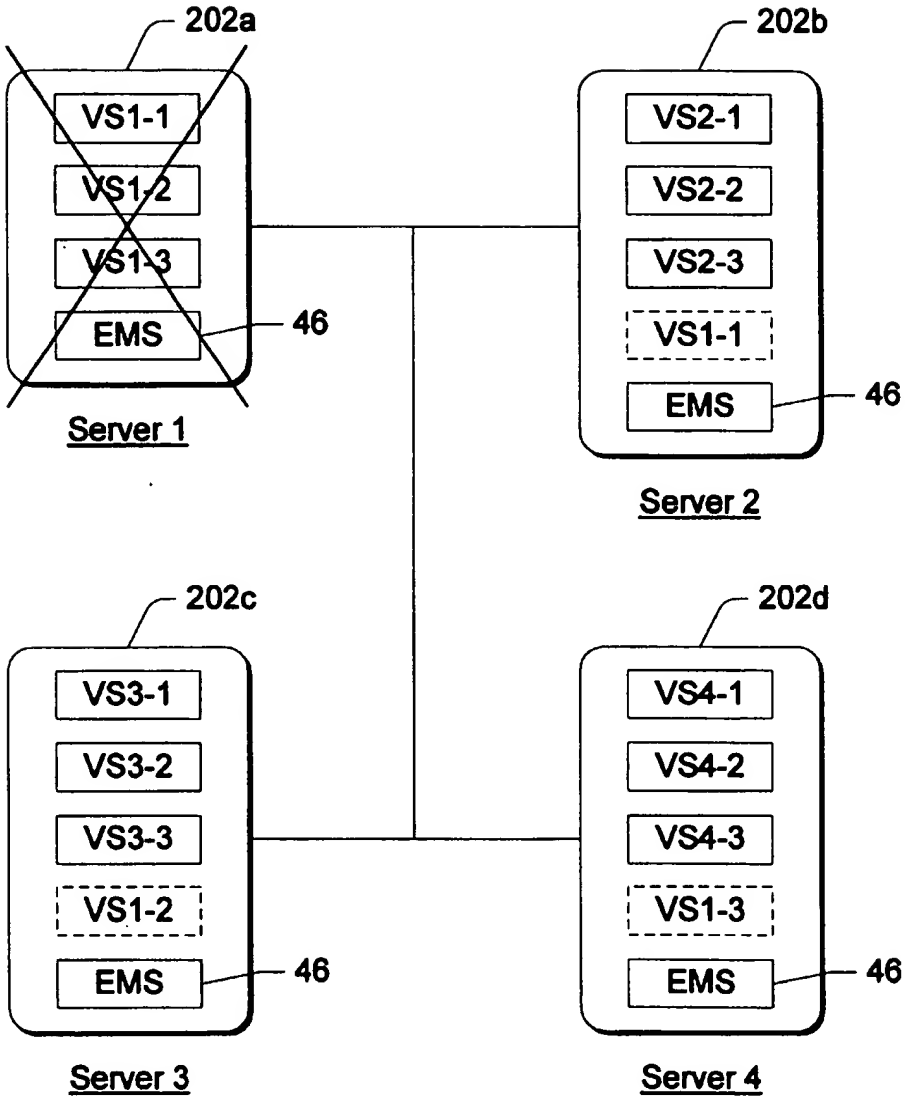

202a
VS1-1
VS1-2
VS1-3
EMS
46
Server 1
202b
VS2-1
VS2-2
VS2-3
VS1-1
EMS
46
Server 2
202c
VS3-1
VS3-2
VS3-3
VS1-2
EMS
46
Server 3
202d
VS4-1
VS4-2
VS4-3
VS1-3
EMS
46
Server 4

Fig. 8

NETWORK SERVER SUPPORTING MULTIPLE INSTANCE OF SERVICES TO OPERATE CONCURRENTLY BY HAVING ENDPOINT MAPPING SUBSYSTEM FOR MAPPING VIRTUAL NETWORK NAMES TO VIRTUAL ENDPOINT IDS

TECHNICAL FIELD

This invention relates to host network servers for computer network systems. More particularly, this invention relates to an endpoint mapping subsystem that maps virtual network names in requests destined to virtual services on a network server into predefined virtual endpoint IDs associated with the virtual services so that one physical server can support multiple instances of the same service. This allows the service to migrate from server to server and thereby improve the service's availability and manageability.

BACKGROUND OF THE INVENTION

A host network server serves data to one or more client computers over a network. The server typically runs one or more services, each of a service type, which can be used by the clients. For example, the server might support the following service types email, web server, database, and the like. Each service is assigned a unique designated endpoint so that clients communicate with a particular service on the host computer by sending requests to, and receiving replies from, the designated endpoint for the particular service.

A client connects to a particular service, via a protocol, by specifying a locator ID and an endpoint ID that are known to both the client and the host server. Examples of common protocols include the named pipe protocol (NPP), the Transmission Control Protocol/Internet Protocol (TCP/IP), and the User Datagram Protocol/Internet Protocol (UDP/IP). For TCP/IP and UDP/IP, the locator ID is an Internet Host Address (or IP Address), and the endpoint ID is a port ID. For NPP, the locator ID is a NetBIOS name, and the endpoint ID is a named pipe name.

The endpoint ID is used to identify a particular endpoint on a machine. An endpoint is an object oh the server affected by remote operations from clients and local operations on servers. A named pipe is one example of an endpoint. A temporary file (/tmp/foo) is another example of an endpoint.

A common way for clients and services to agree on locator and endpoint IDs for a service (given a certain protocol) is to use the locator ID of the machine that the service runs on, and to use an endpoint ID permanently associated with the service type. This works because usually one service of a given service type runs per machine. As an example of permanently associating an endpoint ID with a service type, SMTP (Simple Mail Transfer Protocol) runs over TCP/IP and hardwires port **25** as the endpoint ID. As another example, clients of Microsoft's SQL Server 6.5 use the named pipe protocol and assume "\pipe\sq\query" as the endpoint ID. Even when using TCP/IP where IP addresses can be specified, protocols layered above it may not provide the option to specify the IP address (e.g., DCOM RPC (distributed component object model—remote procedure call) with TCP/IP bindings). A port can be supplied, but the address listened on is always "INADDR_ANY".

On the client site, the locator ID and endpoint ID are unique, and the endpoint ID is typically hardwired. This hardwiring is done to make things easier for the user. The user only has to specify a machine name.

Implicit in the conventional processes is the assumption that each machine will only be running one instance of the service. However, current server technology allows administrators to run multiple instances of the same service on a single machine. For example, a database administrator might wish to run multiple instances of the SQL database service so that more physical memory can be used than can be addressed by a single address space. In this case, one instance of SQL covers one address space and another instance of SQL covers another address space. To the client, each SQL instances functions as its own service running on its own machine. In this manner, the physical host server can be said to support multiple "virtual services" on multiple "virtual servers".

As another example, it is not uncommon for a Web server to support thousands of domains on the same Web service. To the client, however, each domain functions as its own service as if running on its own HTTP (Hypertext Transfer Protocol) server on its own machine. Here again, one physical host server is effectively running multiple "virtual services" on multiple "virtual servers".

A fault tolerant system in computer clustering is another context in which running multiple instances of the same service is desired. A computer cluster is made up of multiple computers interconnected by a network. Services, and data managed by the services, are distributed across the various computers in an effort to balance the load placed on any one computer. A fault tolerant system is designed into the computer cluster to accommodate failure of a computer, or component therein, within the cluster. When a computer fails, the services and resources handled by the failed computer are shifted to one or more other computers in the cluster to prevent loss of services to the clients. In this setting, there may arise a need to run multiple instances of a service on the same host machine. In the database arena, for example, a computer cluster configured to run multiple SQL services across multiple machines might be forced, due to a failure of one or more machines, to run multiple instances of the SQL service on one machine.

Unfortunately, inning multiple instances of the same service on a single machine creates a problem in how the clients connect to the instances. When multiple instances live on one machine, the endpoint IDs are no longer unique. Both the locator ID and the endpoint ID need to be considered to uniquely identify the instance of the service. However, in some API's that use the protocols (e.g., named pipe interface) the locator ID is stripped away because the location is assumed to be "here" at the local computer.

In addition, many server operating systems do not allow more than one instance of a service to bind to the designated endpoint for that service. As a result, other instances remain idle until they are bound to the endpoint. For example, such operating systems such as Windows NT operating system from Microsoft Corporation and the OS/2 operating system from IBM do not permit more than one instance of a service to bind to the designated named pipe for that service. Since only ode instance of a service is bound to the named pipe at any one time, the other instances are left idle, waiting for their opportunity to bind to the named pipe.

Abandoning the conventional protocols is not an option. Many existing client applications across many diverse platforms connect to services using the conventional protocols. Eliminating the protocols would require rewriting the client applications.

Accordingly, there is a need to improve server operating systems to enable multiple instances of a service on the same machine to bind concurrently to endpoints of the client-server protocols.

SUMMARY OF THE INVENTION

This invention concerns a protocol endpoint mapping system, which is implemented on a host server that serves clients over a network. The host server supports one or more services, each of some service type. Example service types are Microsoft SQL Server, Microsoft Exchange Server, and FTP (File Transport Protocol) server. A service is an independent instance of a service type, which may be running or not running. A client connects to a particular service, via a protocol, by specifying a locator ID and an endpoint ID that are known to both the client and the host server. Examples of such protocols include the named pipe protocol (NPP), the Transmission Control Protocol/Internet Protocol (TCP/IP), and the User Datagram Protocol/Internet Protocol (UDP/IP). Each service is assigned a unique designated endpoint so that clients communicate with a particular service on the host computer by sending requests to, and receiving replies from, the designated endpoint for the particular service.

The host server supports multiple virtual services, which are multiple instances of a single service type. The clients can individually request the virtual services by using virtual network names assigned to the virtual services. The virtual network names include a unique locator ID (e.g., server name, IP address) and the designated end point ID (e.g., named pipe name, port ID) that is specified for the service type.

The protocol endpoint mapping subsystem receives requests from the clients and maps the virtual network names in the requests to virtual endpoint IDs established for the virtual services. The virtual endpoint IDs are different than the designated endpoint ID for the generic service type and are thereby used to differentiate among the virtual services.

According to one implementation, the endpoint mapping subsystem is incorporated into an operating system of the network server. In another implementation, the endpoint mapping subsystem is constructed as a dynamic-link library that is callable by the service.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 shows a software architectural representation of a computer network system.

FIG. 2 is a block diagram of a server computer used in the computer network system.

FIG. 3 is a block diagram showing an endpoint mapping subsystem implemented in software as part of a server operating system on the host computer of FIG. 2.

FIG. 4 is a flow diagram showing steps in a method for creating a virtual network name for a virtual service.

FIG. 5 is a flow diagram showing steps in a method for creating new endpoint IDs.

FIGS. 6*a*, 6*b*, 6*c* and 6*d* show various table-based data structures used to correlate locator IDs and designated endpoint IDs with a new virtual endpoint ID.

FIG. 7 is a flow diagram showing steps in a method for mapping client requests for virtual services from a designated endpoint to relative endpoints set up for the virtual services.

FIG. 8 shows a computer cluster in which the endpoint mapping subsystem of FIG. 3 is used in the context of fault tolerant systems.

DETAILED DESCRIPTION

FIG. 1 shows a software architectural representation of a computer network system **20**. The network system **20** has a host network server **22** connected to serve multiple clients **24(1)** and **24(2)** over a network **26**. The network **26** is representative of many diverse network technologies (e.g., Ethernet, satellite, modem-based, etc.) and different configurations, including a LAN (local area network), a WAN (wide area network), and the Internet. For discussion purposes, the computer network system **20** is described in the context of a database system in which the network server **22** is a database server that provides database services to the clients **24(1)** and **24(2)** over the network **26**. It is noted, however, that this invention is not limited to the database context, but may be implemented in other contexts.

Each client computer **24(1)** and **24(2)** has a network layer **28(1)** and **28(2)** that facilitates data transfer over the network **26**. Each client also has an operating system (OS) kernel **30(1)** and **30(2)**. As an exemplary implementation, the client OS kernels **30(1)** ant **30(2)** are implemented as a Windows brand operating system from Microsoft Corporation, such as Windows 3.1, Windows 95, Windows 98, Windows NT Workstation, and other Windows derivatives. The client computers **24(1)** and **24(2)** are running SQL applications **32(1)** and **32(2)** to enable access to the SQL server database maintained at the network server **22**. The client computers may be implemented, for example, as microprocessor-based general purpose computers.

Network serve; **22** has a network layer **38** and an operating system kernel **40** atop the network layer. As one example, the operating system **40** is the Windows NT Server operating system from Microsoft Corporation, although other types of server operating systems can be used.

One or more services are supported by the server operating system **40**, as represented in FIG. **1** by service **42**. The services are independent instances of particular service types, such as Microsoft SQL Server, Microsoft Exchange Server, and FTP (File transport Protocol) server. In the continuing example, the service type is a SQL database service, although other exemplary types of services may be used.

The clients **24(1)** and **24(2)** connect to a particular service, via a protocol, by specifying a locator ID and an endpoint ID that are known to both the clients and the host server **22**. Examples of such protocols include the named pipe protocol (NPP), the Transmission Control Protocol/Internet Protocol (TCP/IP), and the User Datagram Protocol/Internet Protocol (UDP/IP). Each service **42** is assigned a unique designated endpoint so that the clients **24(1)** and **24(2)** communicate with a particular service on the host computer by sending requests to, and receiving replies from, the designated endpoint for the particular service. The following table summarizes exemplary protocols and the associated endpoints, endpoint IDs, and locator IDs for each protocol.

Protocol	Endpoint	Endpoint ID	Locator ID
Named Pipe	Named Pipe Queue	Named Pipe Name	Server Name
TCP/IP	Server-side Socket	Port ID	IP Address
UDP/IP	Server-side Socket	Port ID	IP Address

In the exemplary illustration, two instances **42(1)** and **42(2)** are concurrently executing on the host server **22**. Each instance **42(1)** and **42(2)** presents itself to the clients **24(1)** and **24(2)** as a "virtual service". Even though the virtual services are instances of the same service type on the same machine, each virtual service appears to the client as its own service running on its own machine. Hence, the service **42**

is said to support multiple "virtual services"; or said another way, the host server **22** is said to present multiple "virtual servers".

Generally speaking, the operating system **40** utilizes a protocol construction to handle client requests. The operating system **40** maintains pre-known designated endpoints for the various services in an endpoint file system (EFS) **44**. The server operating system **40** also implements an endpoint mapping subsystem (EMS) **46** that enables incoming requests for virtual services to be mapped from the designated endpoint for that service to other predefined endpoints. The predefined endpoints for the virtual services **42(1)** and **42(2)** are different than the designated endpoint for the generic service **42**. It is noted, however, that the endpoint mapping subsystem **46** may alternatively be implemented as a dynamic-link library (DLL) **48** that is callable by the virtual service **42(1)**, and performs the endpoint mapping prior to reaching the virtual service **42(1)**.

Prior to this invention, the operating system used the same designated endpoint for different virtual services. As a result, only one request is handled at the endpoint at a time. Other requests remain idle in a queue and unanswered until the endpoint is available again. More specifically, request packets arriving at the host server specify a locator ID and an endpoint ID. The protocol layer (e.g., TCP) strips off the locator ID since it assumes that the host is only running one service locally that is specified by the endpoint ID. The endpoint ID is passed up to the next higher layer in the protocol stack for use in finding the service.

To illustrate this problem with in a specific context, consider the named pipe protocol. The server operating system **40** maintains well-known designated named pipes for the various services in a named pipe file system. Each named pipe has a unique name that distinguishes it from other named pipes. A pipe name that is accessed remotely over a network typically takes on the following form:

```
\server\pipe\service_name
```

where "server" is replaced with the real or virtual server name and "service_name" is replaced with the name of the service. In FIG. **1**, the server **22** supports two virtual SQL services. The clients **24(1)** and **24(2)** address the virtual SQL services **42(1)** and **42(2)** using the following virtual network names:

```
\virtual_server1\pipe\SQL\query
\virtual_server2\pipe\SQL\query
```

The server operating system **40** inherently assumes that only one of the instances **42(1)** and **42(2)** at a time can bind to the designated named pipe for the service **42**. In this case, the generally designated named pipe is "\pipe\SQL\query". When the requests for the different virtual services are handled, the "virtual_serve" portion of the name is stripped off at a low protocol layer as being immaterial. That is, the operating system **40** resolves different virtual network names to the same named pipe name. Thus, the above virtual network names for different virtual services are typically (in the absence of this invention) resolved to the identical named pipe:

```
\\.\pipe\SQL\query
```

Although the problem has been described in the context of the named pipe protocol, similar problems exist for other protocols, such as TCP/IP and UDP/IP. In TCP/IP or UDP/IP, a client request typically contains a port ID and an IP address to designate a; particular virtual service. The service type for the virtual service is typically assigned just one corresponding port ID at the host computer. When the request is received, the TCP layer strips off the IP address as unnecessary since it is designating a service running on the local machine, thereby leaving only the port ID. Because the operating system uses the same designated port ID for a service type, the same port ID is applied to all of the virtual services, allowing only one virtual service to bind to the port at a time. Requests destined for other virtual services remain idle and unanswered until the port is available again and can bind to the other virtual service.

To avoid this problem, the endpoint mapping subsystem **46** maps incoming requests for virtual services from a designated endpoint for that service to other predefined endpoints. The predefined endpoints for the virtual services **42(1)** and **42(2)** are different than the designated endpoint for the generic service **42**. The operating system **40** then binds the endpoints to the various instances **42(1)** and **42(2)** of the same service **42**. As a result, the server can handle concurrent access to multiple virtual services.

More particularly, the client request includes a locator ID and an endpoint ID. The mapping subsystem **46** uses the locator ID to identify the virtual service and maps the endpoint ID (which is general for the service type) to a new ID associated with the vital service.

Consider the endpoint mapping subsystem **46** in the context of various protocols. For named pipe protocol, the client request contains a server name (e.g., virtual_server1) and a named pipe name (e.g., pipe\SQL\query). The mapping subsystem **46** replaces the virtual network name of a virtual service (as used by the clients **24(1)** and **24(2)**) with named pipe names that are used by the server **22**. As one implementation, the mapping subsystem **46** is configured to insert the virtual network name string into the named pipe name. As an example, the virtual network names for virtual SQL services are mapped as follows:

Virtual Endpoint	Map to. . .	New Endpoint
\virtual_server1\pipe\SQL\query	→	\pipe\$$\virtual_server1\SQL\query
\virtual_server2\pipe\SQL\query	→	\pipe\$$\virtual_server2\SQL\query

Once mapped, the mapping subsystem **46** forwards the client request to the file system **44** using the new named pipe name. The virtual service then handles the request using the new named pipe, as opposed to the named pipe that is designated for the primary service.

For TCP/IP or UDP/IP, the client request contains an IP address and a port ID (which is general to the service type). The mapping subsystem **46** uses the IP address to identify the particular virtual service and then maps the port ID

contained in the request to a new port ID. For instance, suppose the IP address is "129.56.85.7" for one virtual service and "129.56.85.7" for another virtual service. These virtual services are instances of the service type having a designated port ID "80".

Virtual Endpoint	Map to. . .	New Endpoint
129.56.85.7:80	→	129.56.85.7:81
129.56.85.8:80	→	129.56.85.8:82

FIG. **2** shows an example implementation of a host server computer **22** in more detail. The server **22** is a general purpose computing device in the form of a conventional personal computer that is configured to operate as a host network server. The server computer **22** includes a processing unit **60**, a system memory **62**, and a system bus **64** that couples various system components including the system memory **62** to the processing unit **60**. The system bus **64** may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory **62** includes read only memory (ROM) **66** and random access memory (RAM) **68**. A basic input/output system **70** (BIOS) is stored in ROM **66**.

The server computer **22** also has one or more of the following drives: a hard disk drive **72** for reading from and writing to a hard disk, a magnetic disk drive **74** for reading from or writing to a removable magnetic disk **76**, and an optical disk drive **78** for reading from or writing to a removable optical disk **80** such as a CD ROM or other optical media. The hard disk drive **72**, magnetic disk drive **74**, and optical disk drive **78** are connected to the system bus **64** by a hard disk drive interface **82**, a magnetic disk drive interface **84**, and an optical drive interface **86**, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer readable instructions, data structures, program modules and other data for the personal computer.

Although a hard disk, a removable magnetic disk **76**, and a removable optical disk **80** are described, it should be appreciated by those skilled in the art that other types of computer readable media can be used to store data. Other such media include magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), read only memories (ROM), and the like.

A number of program modules may be stored on the hard disk, magnetic disk **76**, optical disk **80**, ROM **66**, or RAM **68**. These programs include the operating system **40**, one or more application programs **90** (such as service **42**), other program modules **92**, and program data **94**. A user may enter commands and information into the personal computer **22** through input devices such as keyboard **96** and pointing device **98**. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices ate often connected to the processing unit **60** through a serial port interface **100** that is coupled to the system bus **64**, but may be connected by other interfaces, such as a parallel port, game port, or a universal serial bus (USB). A monitor **102** or other type of display device is also connected to the system bus **64** via an interface, such as a video adapter **104**. In addition to the monitor, personal computers typically include other peripheral output devices (not shown) such as speakers and printers.

The server computer **22** is connected to the network **26** through a network interface or adapter **106**, a modem **108**, or other means for establishing communications over the Internet. The modem **108**, which may be internal or external, is connected to the system bus **64** via the serial port interface **100**.

FIG. **3** shows an implementation in which the endpoint mapping subsystem **46** is incorporated into the server operating system **40** on the host server computer **22**. The mapping subsystem **46** includes a virtual locator ID creator module **110**, a virtual endpoint ID creator module **112**, a virtual endpoint ID mapper module **114**, and an endpoint assignor module **116**.

The virtual locator ID creator **110** creates new network names that can be dynamically mapped to one of several server machines. For NPP, the locator ID creator generates a virtual server name for the named pipe. For TCP/IP or UDP/IP, the locator ID creator defines new IP addresses.

FIG. **4** shows steps in a method that is implemented by the virtual locator ID creator **110**. At step **120** in FIG. **4**, the server uses input from the user or administrator to define the one or more virtual services to be supported by the server **22**. For instance, an Internet service provider might wish to define various email post offices that are supported by the same email service, or a number of domain names that are supported by the same Web service.

At step **122** in FIG. **4**, the locator ID creator **110** assigns a virtual network name to each of the virtual services. Remote clients use the virtual network names to address requests for the virtual services. It is noted that the locator ID creator **110** may perform this step automatically, or with the assistance of an administrator who assigns network names to the virtual services.

With reference again to FIG. **3**, the virtual endpoint ID creator **112** is responsible for creating new endpoint IDs for the virtual services. There is one virtual endpoint ID creator **112** for service type. The endpoint ID creator **112** generates a new endpoint ID based upon the virtual locator ID assigned to the virtual service and the original default endpoint ID that is designated for the service type. The new endpoint ID can be the same as the designated endpoint ID if one service of the service type is running. However, where two or more virtual services are running, new endpoint IDs are created.

In most cases, the new endpoint ID will be different from the designated endpoint ID for the service type. In the NPP context, the virtual endpoint ID creator **112** defines, for each virtual service, a new named pipe name given the designated named pipe name for the service type and the server name. For TCP/IP or UDP/IP, the endpoint ID creator **112** generates new port IDs from the designated port ID and the IP address.

FIG. **5** shows steps in an exemplary method implemented by the endpoint ID creator module **112** to correlate new endpoint IDs with the original locator IDs and designated endpoint IDs contained in the client requests. At step **130**, the endpoint ID creator **112** constructs a table or other data structure. The table has a first field to hold the locator ID, second field to hold the designated endpoint ID, and a third field to hold the new endpoint ID. When a new endpoint is assigned to an existing endpoint ID, the endpoint ID creator **112** adds a new entry to the table (step **132**). The entry can be deleted from the table when the endpoint is closed.

In another implementation, the mapping is fixed and there is no need for a mapping table. This implementation has the benefit of being more streamlined because the process does not involve lookup operations. However, a fixed mapping

system is not as flexible or adaptable, which is a disadvantage in comparison to the table approach.

FIG. *6a* shows a mapping table **140** or other data structure that stores a new endpoint ID in association with the locator ID and designated endpoint ID from which it is derived. The mapping table **140** has a first field for the locator ID, second field for the designated endpoint ID, and a third field for the new endpoint ID. FIGS. *6b* and *6c* show examples of mapping tables used in the NPP and TCP/IP (or UDP/IP) contexts. In FIG. *6b*, a mapping table **142** is constructed for the named pipe protocol to correlate the server name and designated named pipe name with the new named pipe name to which they map. In FIG. *6c*, a mapping table **144** is arranged for TCP/IP or UDP/IP to correlate the IP address and designated port number with the new port number to which they map. Using these tables, the host can map an incoming destination address to an internal or relative address that correctly routes the requests to the appropriate virtual services.

FIG. *6d* shows another implementation in which a second table **146** is used to determine preliminarily whether the designated endpoint ID is mapped at all. In this example, the second table **146** is used in conjunction with the mapping table **144** from FIG. *6c*. She second table **146** lists the assigned port IDs to determine whether a particular port ID has been mapped to a new port ID. If so, the mapping table **144** is consulted to determine the new port ID.

With reference to FIG. **3**, the virtual endpoint ID mapper **114** maps incoming client requests to the appropriate new endpoint IDs so that the requests are passed to the intended virtual services. When a client request is received at the server **22**, the virtual endpoint ID mapper **114** uses the mapping table **140** to convert from the old locator ID and designated endpoint ID to the virtual endpoint ID for the virtual service to which the request is directed.

The endpoint assignor **116** assigns the virtual endpoint ID to an existing endpoint (e.g., named pipe queue, socket). Another data structure can be configured to associate the virtual endpoint ID with the endpoint, although this may be implicit in the mapping table in that endpoint IDs specify particular endpoints. The endpoints are previously created using existing operating system functions. For example, in the named pipe protocol context, the function call "CreateNamedPipe" forms a queue data structure for a named pipe. The queue data structure implements the pipe structure that passes data to and from a client, either in requests or bytes. In the TCP/IP and UDP/IP contexts, the operating system opens a socket to define the endpoint.

FIG. **7** shows steps in the process for handling client requests, including both instructions executed at the client and instructions executed at the server. The endpoint ID mapper **114** implements the server-side process steps.

At step **150** in FIG. **7**, a client computer submits a request to the server. The request contains a locator ID (e.g., server name, IP address) and an endpoint ID (e.g., named pipe name, port ID). The server evaluates whether the request is of a certain protocol (step **152**). If it is, the server determines whether the endpoint ID contained in the request is mapped to a virtual endpoint ID (step **154**). This step can be performed by looking up the endpoint ID in the second table **146** and determining whether it is mapped or not.

Assuming it is mapped to a virtual endpoint ID (the "yes" branch from step **154**), the server uses the mapping table **140** to determine the virtual endpoint ID associated with the endpoint ID contained in the request (step **156**). The server then converts the request to specify the virtual endpoint ID (step **158**). The request is then passed onto the appropriate virtual service for handling (step **160**).

On the other hand, in the event that the endpoint ID is not mapped to a virtual endpoint ID (the "not" branch from step **154**), the request is handled using the original endpoint ID (step **162**).

After an endpoint and virtual endpoint ID is specified for communication with the client, all subsequent client requests contain the virtual endpoint ID and the server ascertains the endpoint from the virtual endpoint ID. Alternatively, the server may pass a handle back to the client, which can be used to directly locate the endpoint for subsequent communication.

The processes described herein may also be used to accommodate remote service operations, such as starting a process, stopping a process, and other operations that may be initiated remotely from the server. Typically, the server operating system assigns one endpoint to listen to remote service operations. If one client sends a request to stop a virtual service, the operating system interprets that request to stop the service itself (which might be supporting multiple virtual services), rather than; the desired single virtual service. To avoid this problem, the endpoint mapping subsystem can be used to translate service operation requests to the appropriate named pipes set up for the individual virtual services. In this manner, the clients can direct service operation requests to specific virtual services, without affecting other virtual services.

The endpoint mapping subsystem is particularly well suited for fault tolerant systems in computer clusters. When a node of a computer cluster fails, the services supported by the failed node are shifted to another node. In the event that this transfer results in running two versions of the same service, the endpoint mapping subsystem implemented on the non-failed node accommodates client requests to the two different versions.

FIG. **8** shows a computer cluster **200** having four interconnected servers **202***a*–**202***d*. Each sever runs multiple virtual services "VS". Each server is implemented with the endpoint mapping subsystem **46** and a failover system (not shown), both of which may be incorporated into the operating system. Now, assume that one of the servers, say server **202***a*, crashes. As part of the fault tolerant system, the virtual services VS1-1 to VS1-3 from the failed server **202***a* are transferred to the surviving servers **202***b*–**202***d*, as indicated by the dashed blocks located on each of the surviving servers.

When a new client request is received at the cluster **200** following failure of server **202***a*, the failover system routes the request to the appropriate surviving server. This rerouting is conventional and well-known in the failover technology. Once the request for a virtual service VS1-1 reaches the surviving server **202***b*, the endpoint mapping subsystem uses the locator ID and endpoint ID contained in the request to map to a virtual endpoint ID and corresponding endpoint at the surviving server **202***b*. The mapping process is the same as described above.

Although the invention has been described in language specific to structural features and/or methodological steps, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or steps described. Rather, the specific features and steps are disclosed as preferred forms of implementing the claimed invention.

What is claimed is:

1. A network server system for handling requests for a service from one or more clients, the network server system comprising:

a processing unit;

a memory subsystem;

a service stored in the memory subsystem and executable on the processing unit, the service being configured to receive requests from the clients via a designated endpoint ID;

the service supporting multiple virtual services that can be individually requested by the clients using virtual network names for the virtual services; and

an endpoint mapping subsystem to map the virtual network names contained in client requests for the virtual services to virtual endpoint IDs that are different than the designated endpoint ID and are associated with the requested virtual services.

2. A system as recited in claim 1, wherein the endpoint mapping subsystem comprises:

a virtual endpoint ID creator to define the virtual endpoints associated with the requested virtual services; and

a virtual endpoint ID mapper to map the virtual network names to the virtual endpoints associated with the requested virtual services.

3. A system as recited in claim 1, further comprising a network server operating system stored in the memory subsystem and executed on the processing unit, the endpoint mapping subsystem being implemented as part of the network operating system.

4. A system as recited in claim 1, further comprising a dynamic-link library stored in the memory subsystem, the endpoint mapping subsystem being implemented as part of the dynamic-link library.

5. A network server cluster for handling requests from one or more clients, the server cluster having multiple interconnected servers, whereby each server comprises:

a processing unit;

a memory subsystem;

a service stored in the memory subsystem and executable on the processing unit, the service being configured to receive requests from the clients via a designated endpoint ID;

the service supporting multiple virtual services that can be individually requested by the clients using virtual network names for the virtual services;

an endpoint mapping subsystem to map the virtual network names contained in client requests for the virtual services to virtual endpoint IDs that are different than the designated endpoint ID and are associated with the requested virtual services; and

whereby in an event that one of the servers in the network server cluster fails and the viral services on the failed server are transferred to various surviving servers, the endpoint mapping subsystem mapping the virtual network names contained in the client requests for the transferred virtual services to virtual endpoint IDs used at the surviving servers to locate the transferred virtual services.

6. An endpoint mapping component for use in a network server, the network server prodding a service that receives requests from clients via a designated endpoint ID, the service supporting multiple virtual services that can be individually requested by the clients using virtual network names assigned to the virtual services, the Endpoint mapping component comprising:

a virtual endpoint ID creator to define a virtual endpoint ID for a virtual service that is requested by a client, the virtual endpoint ID being different from the designated endpoint ID, the virtual service having a corresponding virtual network name that is contained in the client request; and

a virtual endpoint ID mapper to change the virtual network name of the virtual service to the virtual endpoint ID that is created by the virtual endpoint ID creator for the requested virtual service so that the client request containing the virtual network name is passed to the requested virtual service.

7. A named pipe mapping component as recited in claim 6, wherein the virtual endpoint ID creator makes an entry in a data structure that associates the virtual endpoint ID with the designated endpoint ID.

8. A named pipe mapping component as recited in claim 6, further comprising a data structure to associate the virtual endpoint ID with the designated endpoint ID.

9. A named pipe mapping component as recited in claim 6, wherein the virtual network name includes a locator ID and the designated endpoint ID, and further comprising a data structure to associate the virtual endpoint ID with the locator ID and the designated endpoint ID.

10. A network server operating system embodied on a computer-readable medium comprising the endpoint mapping component as recited in claim 6.

11. A dynamic-link library embodied on a computer-readable medium comprising the endpoint mapping component as recited in claim 6.

12. A virtual endpoint ID creator module embodied as a computer program on a computer-readable medium, the virtual endpoint ID creator module being implemented in a network server connected to serve one or more clients over a computer network system, the network server providing a service that receives requests from the clients via a designated endpoint ID, the service supporting multiple virtual services that can be individually requested by the clients, the virtual endpoint ID creator module comprising:

first code means for creating a virtual endpoint ID based at least in part on the designated endpoint ID for use with a corresponding virtual service; and

second code, means for associating the virtual endpoint ID with the designated endpoint ID so that receipt of a client request specifying the designated endpoint ID can be mapped to the virtual endpoint ID for handling by the corresponding virtual service.

13. A network server operating system embodied on a computer-readable medium comprising the virtual endpoint ID creator module as recited in claim 12.

14. A dynamic-link library embodied on a computer-readable medium comprising the named pipe creator module as recited in claim 12.

15. A virtual endpoint ID mapper module embodied as a computer program on a computer-readable medium, the module being implemented in a network server connected to serve one or more clients over a computer network system, the network, server providing a service that receives requests from the clients via a designated endpoint ID, the service supporting multiple virtual services that can be individually requested by the clients, the virtual endpoint ID mapper module comprising:

first code means for ascertaining a predefined virtual endpoint ID that is associated with a virtual network name contained in the client request for a particular virtual service; and

second code means for using the virtual endpoint ID to locate the particular virtual service.

16. A virtual endpoint ID mapper module as recited in claim 15, further comprising third code means for looking up the virtual endpoint ID in a table that correlates the virtual endpoint ID with the virtual network name.

17. A virtual endpoint ID mapper module as recited in claim 15, wherein the virtual endpoint ID specifies a corresponding endpoint, further comprising third code means for returning a handle for the endpoint to the client making the request.

18. A network server operating system embodied on a computer-readable medium comprising the virtual endpoint ID mapper module as recited in claim 15.

19. A dynamic-link library embodied on a computer-readable medium comprising the virtual endpoint ID mapper module as recited in claim 15.

20. A network server operating system for execution on a network server that is configured to serve one or more clients over a computer network system, the network server providing a service that receives requests from the clients via a designated endpoint ID, the service supporting multiple virtual services that can be individually requested by the clients using virtual network names for the virtual services, the virtual network names containing in part the designated endpoint ID, the network server operating system being implemented with an endpoint mapping subsystem to map the virtual network names in the client requests to predefined virtual endpoint IDs that are different than the designated endpoint ID and are associated with the requested virtual services.

21. A network operating system as recited in claim 20, wherein the endpoint mapping subsystem establishes an endpoint that corresponds to the virtual endpoint ID.

22. A network operating system as recited in claim 20, wherein the endpoint mapping subsystem establishes a data structure that correlates the virtual endpoint IDs with the virtual network names.

23. In a network server having a service that receives requests from clients via a designated endpoint ID, the service supporting multiple virtual services that can be individually requested by the clients, a method comprising the following steps:

creating a virtual endpoint ID for a virtual network name associated with a particular virtual service;

associating the virtual endpoint ID with the virtual network name; and

in response to a client request for the virtual service, mapping the virtual network name contained in the client request to the virtual endpoint ID for the virtual service.

24. A method as recited in claim 23, wherein the creating step comprises the step of correlating the virtual endpoint ID with the virtual network name in a data structure.

25. A method as recited in claim 23, wherein the virtual network name includes a locator ID and the designated endpoint ID, further comprising the step of correlating the virtual endpoint ID with the locator ID and the designated endpoint ID using a lookup table structure.

26. A computer-readable medium having computer-executable instructions for performing the steps in the method as recited in claim 23.

27. In a network server having a service that receives requests from clients via a designated endpoint ID, the service supporting multiple virtual services that can be individually requested by the clients using virtual network names assigned to the virtual services, a method comprising the following steps:

creating a virtual endpoint ID based at least in part on the designated endpoint ID for use With a corresponding virtual service; and

associating the virtual endpoint ID with the designated endpoint ID so that receipt of a client request specifying the designated endpoint ID can be mapped to the virtual endpoint ID for handling by the corresponding virtual service.

28. A method as recited in claim 27, wherein the creating step comprises the step of correlating the virtual endpoint ID with the virtual network name in a data structure.

29. A computer-readable medium having computer-executable instructions for performing the steps in the method as recited in claim 27.

30. In a network server having a service that receives requests from clients via a designated endpoint ID, the service supporting multiple virtual services that can be individually requested by the clients using virtual network names assigned to the virtual services, a method comprising the following steps:

ascertaining a predefined virtual endpoint ID that is associated with a virtual network name contained in the client request for a particular virtual service; and

using the virtual endpoint ID to locate the particular virtual service.

31. A computer-readable medium having computer-executable instructions for performing the steps in the method as recited in claim 30.

32. In a network server cluster having multiple servers, each server supporting at least one service that receives requests from clients via a designated endpoint ID, the service supporting multiple virtual services that can be individually requested by the clients, wherein in an event that one server in the cluster fails and the virtual services on the failed server are transferred to surviving servers, a method comprising the following steps:

creating a virtual endpoint ID for a virtual network name associated with a transferred virtual service, wherein the transferred virtual service originally resided on the failed server but now resides on one of the surviving servers;

associating the virtual endpoint ID with the virtual network name; and

in response to receiving a client request for the transferred virtual service at the one surviving server, mapping the virtual network name contained in the client request to the virtual endpoint ID for the transferred virtual service.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,247,057 B1
DATED : June 12, 2001
INVENTOR(S) : Barrera, III

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 1,
Line 41, change "oh" to -- on --.

Column 2,
Line 56, change "ode" to -- one --.

Column 4,
Line 17, change "ant" to -- and --.
Line 26, change "serve" to -- server --.
Line 35, change "transport" to -- Transport --.

Column 6,
Line 38, change "vital" to -- virtual --.

Column 7,
Line 58, change "ate" to -- are --.

Column 9,
Line 23, change "She" to -- The --.

Column 10,
Line 2, change "not" to -- no --.

Column 14,
Line 9, change "With" to -- with --.

Signed and Sealed this

Fifteenth Day of January, 2002

Attest:

Attesting Officer

JAMES E. ROGAN
Director of the United States Patent and Trademark Office

US006430622B1

(12) **United States Patent**
Aiken, Jr. et al.

(10) **Patent No.: US 6,430,622 B1**
(45) **Date of Patent: Aug. 6, 2002**

(54) **METHODS, SYSTEMS AND COMPUTER PROGRAM PRODUCTS FOR AUTOMATED MOVEMENT OF IP ADDRESSES WITHIN A CLUSTER**

(75) Inventors: **John Andrew Aiken, Jr.**, Raleigh, NC (US); **Michael Edward Baskey**, Wappingers Falls, NY (US); **James L. Hall**, Raleigh, NC (US); **Dilip Dinkar Kandlur**, Yorktown Heights, NY (US); **Andrew H. Richter**, Raleigh, NC (US)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/401,419**

(22) Filed: **Sep. 22, 1999**

(51) **Int. Cl.**[7] **G06F 15/16**; G06F 15/173

(52) **U.S. Cl.** **709/245**; 709/239; 709/230

(58) **Field of Search** 709/245, 239, 709/230, 232, 217, 105; 714/4

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,403,286 A	9/1983	Fry et al.	364/200
4,495,570 A	1/1985	Kitajima et al.	364/200
4,577,272 A	3/1986	Ballew et al.	364/200

(List continued on next page.)

OTHER PUBLICATIONS

A. Dahlin, et al. EDDIE A Robust and Scalable Internet Server. Ericsson Telecom AB, Stockholm, Sweden, pp. 1–7 (May 1998).
Brochure entitled, ACEdirector™ 8–Port 10/100 MBPS Ethernet Switch. Alteon WebSystems, San Jose, CA (1999).
Brochure entitled, Enhancing Web User Experience with Global Server Load Balancing. Alteon WebSystems, San Jose, CA (Jun. 1999).
Brochure entitled, The Next Step in Server Load Balancing. Alteon WebSystems, San Jose, CA (Nov. 1999).
Mac Devine. Presentation entitled, TCP/IP Application Availability and Workload Balancing in the Parallel Sysplex. Share Technical Conference (Aug. 22–27, 1999).
http://w3.enterlib.ibm.com:80/cgi–bin/bookmgr/books/F1AF7001/1.3.1.2; 1.3.1.2.5 Virtual IP Addressing (VIPA); Excerpt from IP Configuration for OS/390, pp. 1–4 (1998).
http://w3.enterlib.ibm.com:80/cgi–bin/bookmgr/books/F1AF7001/1.3.2; 1.3.20 Device and Link Statement—Virtual Devices (VIPA); Excerpt from IP Configuration for OS/390, pp. 1–3 (1998).
http://w3.enterlib.ibm.com:80/cgi–bin/bookmgr/books/F1AF7001/1.3.2; 1.3.23. Home Statement; Excerpt from IP Configuration for OS/390, pp. 1–6 (1998).
Dahlin, A., et al., EDDIE, A Robust and Scalable Internet Server, Ericsson Telecom AB, Stockholm Sweden, pp. 1–7 (May 1998).

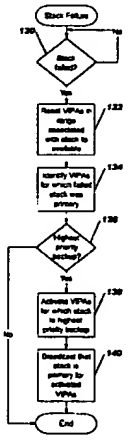
* cited by examiner

Primary Examiner—Zarni Maung
(74) *Attorney, Agent, or Firm*—Jerry W. Herndon; Myers Bigel Sibley & Sajovec, P.A.

(57) **ABSTRACT**

Methods, systems and computer program products are provided for transferring a Virtual IP Address (VIPA) from a first application instance to a second application instance executing on a cluster of data processing systems having a plurality of communication protocol stacks. A list of dynamic VIPAs is distributed among the protocol stacks and a hierarchy of backup communication protocol stacks for the dynamic VIPAs is determined based on the list of dynamic VIPAs. Upon receiving notification of failure of the first stack the second stack evaluates the hierarchy of backup stacks to determine if it is the next stack in the hierarchy for the VIPA associated with the first application instance. If so, then the VIPA associated with the first application instance is transferred to the second communication protocol stack associated with the second application instance.

32 Claims, 9 Drawing Sheets



U.S. PATENT DOCUMENTS

5,031,089 A	7/1991	Liu et al.	364/200
5,675,739 A	10/1997	Eilert et al.	395/200
5,917,997 A	6/1999	Bell et al.	395/182.02
5,923,854 A	7/1999	Bell et al.	395/200.73
5,935,215 A	8/1999	Bell et al.	709/239
5,951,650 A	9/1999	Bell et al.	709/238
6,023,734 A *	2/2000	Ratcliff et al.	709/300
6,078,964 A *	6/2000	Ratcliff et al.	709/300
6,084,859 A *	7/2000	Ratcliff et al.	370/252
6,101,543 A *	8/2000	Alden et al.	709/229
6,175,874 B1 *	1/2001	Imai et al.	709/238
6,185,218 B1 *	2/2001	Raycliff et al.	370/405
6,266,335 B1 *	7/2001	Bhaskaran	370/399
6,343,322 B2 *	1/2002	Nagami et al.	709/227

FIG. 1

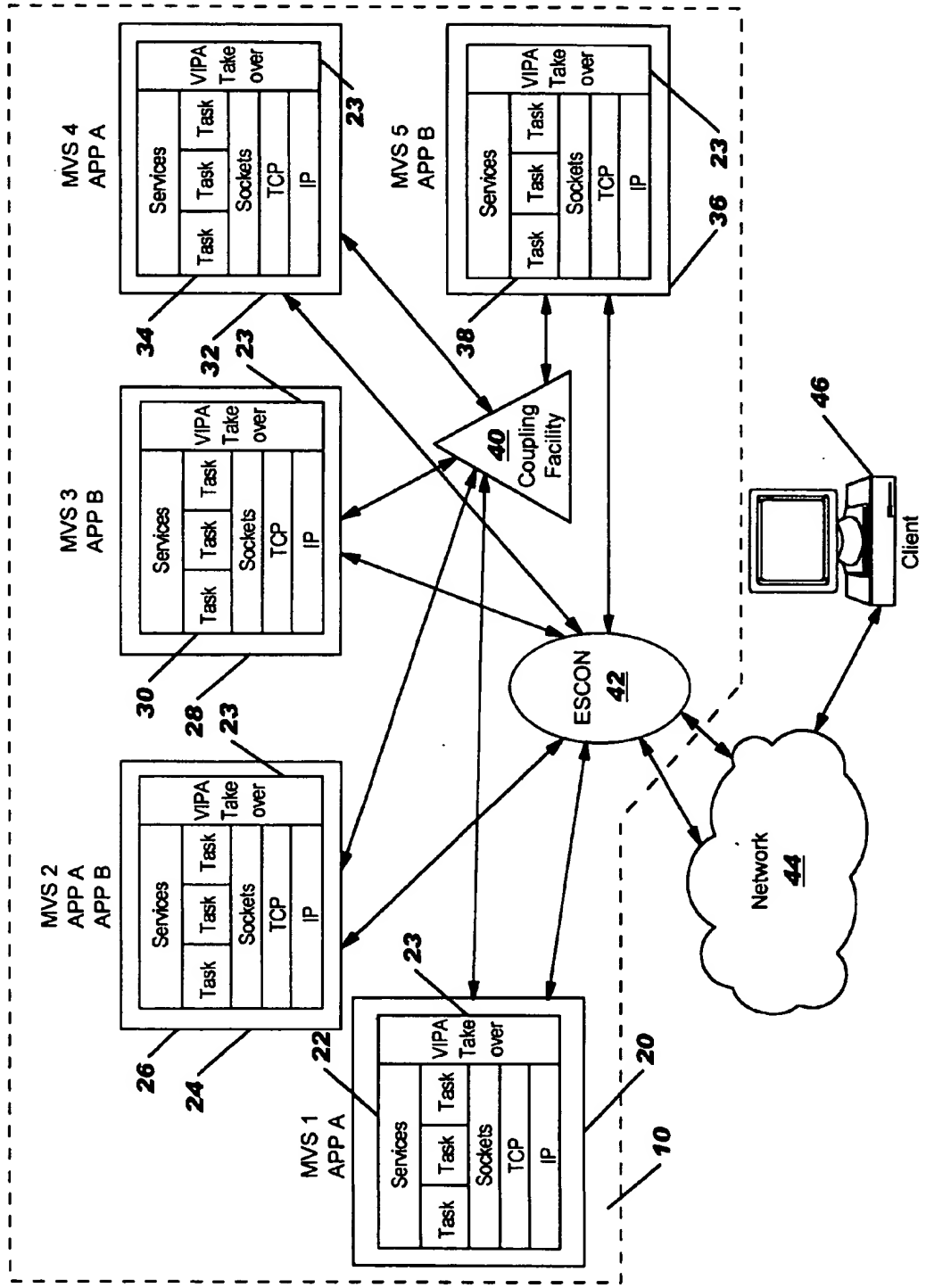

MVS 4
APP A
MVS 5
APP B
MVS 3
APP B
MVS 2
APP A
APP B
MVS 1
APP A
Services
Task
Sockets
TCP
IP
VIPA Take over
23
34
32
38
36
30
28
26
24
22
20
10
40
Coupling Facility
ESCON
42
Network
44
46
Client

FIG. 2

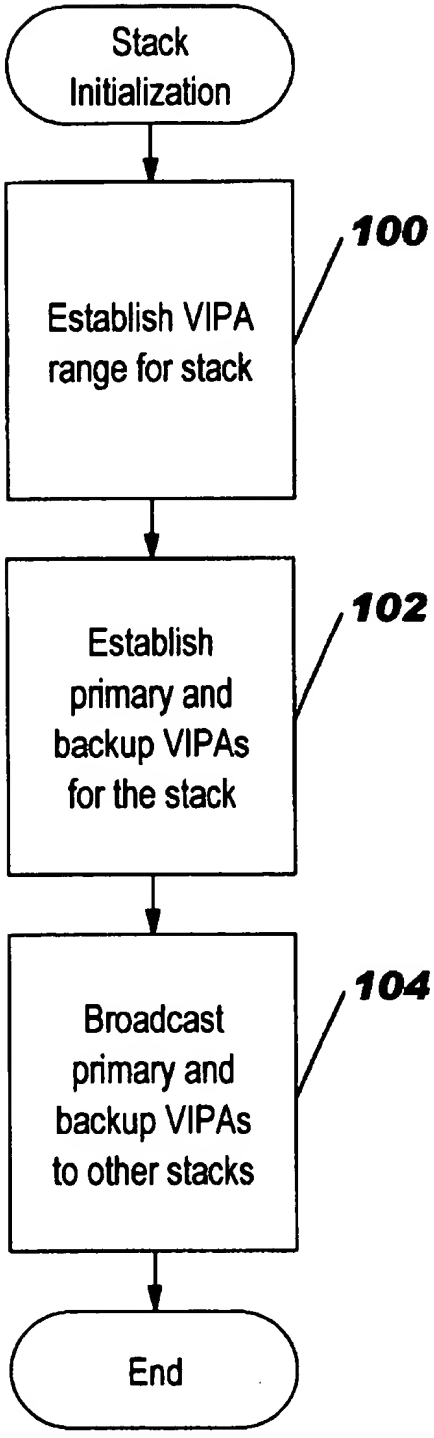

Stack Initialization
Establish VIPA range for stack
100
Establish primary and backup VIPAs for the stack
102
Broadcast primary and backup VIPAs to other stacks
104
End

FIG. 3

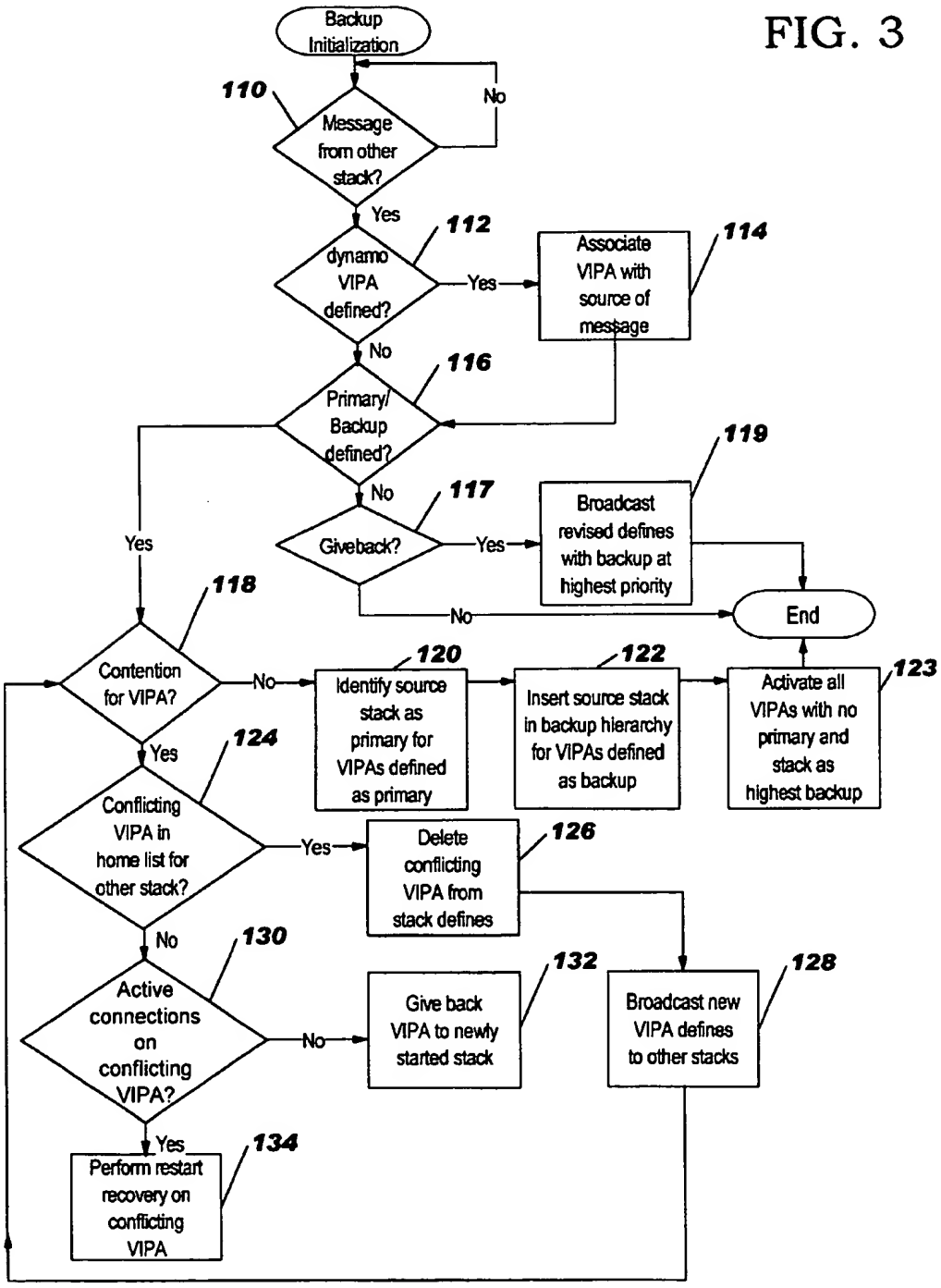

Backup Initialization
110
Message from other stack?
No
Yes
112
dynamo VIPA defined?
Yes
Associate VIPA with source of message
114
No
116
Primary/ Backup defined?
No
117
Giveback?
Yes
Broadcast revised defines with backup at highest priority
119
Yes
No
End
118
Contention for VIPA?
No
120
Identify source stack as primary for VIPAs defined as primary
122
Insert source stack in backup hierarchy for VIPAs defined as backup
Activate all VIPAs with no primary and stack as highest backup
123
Yes
124
Conflicting VIPA in home list for other stack?
Yes
Delete conflicting VIPA from stack defines
126
No
130
Active connections on conflicting VIPA?
No
132
Give back VIPA to newly started stack
128
Broadcast new VIPA defines to other stacks
Yes
134
Perform restart recovery on conflicting VIPA

FIG. 4

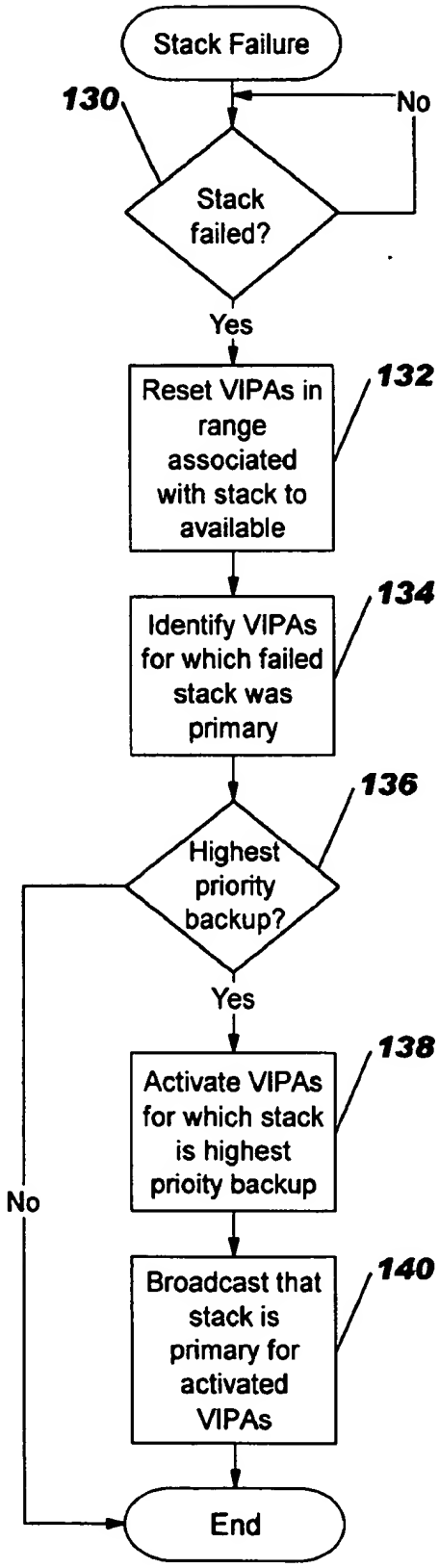

Stack Failure
130
No
Stack failed?
Yes
Reset VIPAs in range associated with stack to available
132
Identify VIPAs for which failed stack was primary
134
Highest priority backup?
136
Yes
No
Activate VIPAs for which stack is highest prioity backup
138
Broadcast that stack is primary for activated VIPAs
140
End

FIG. 5

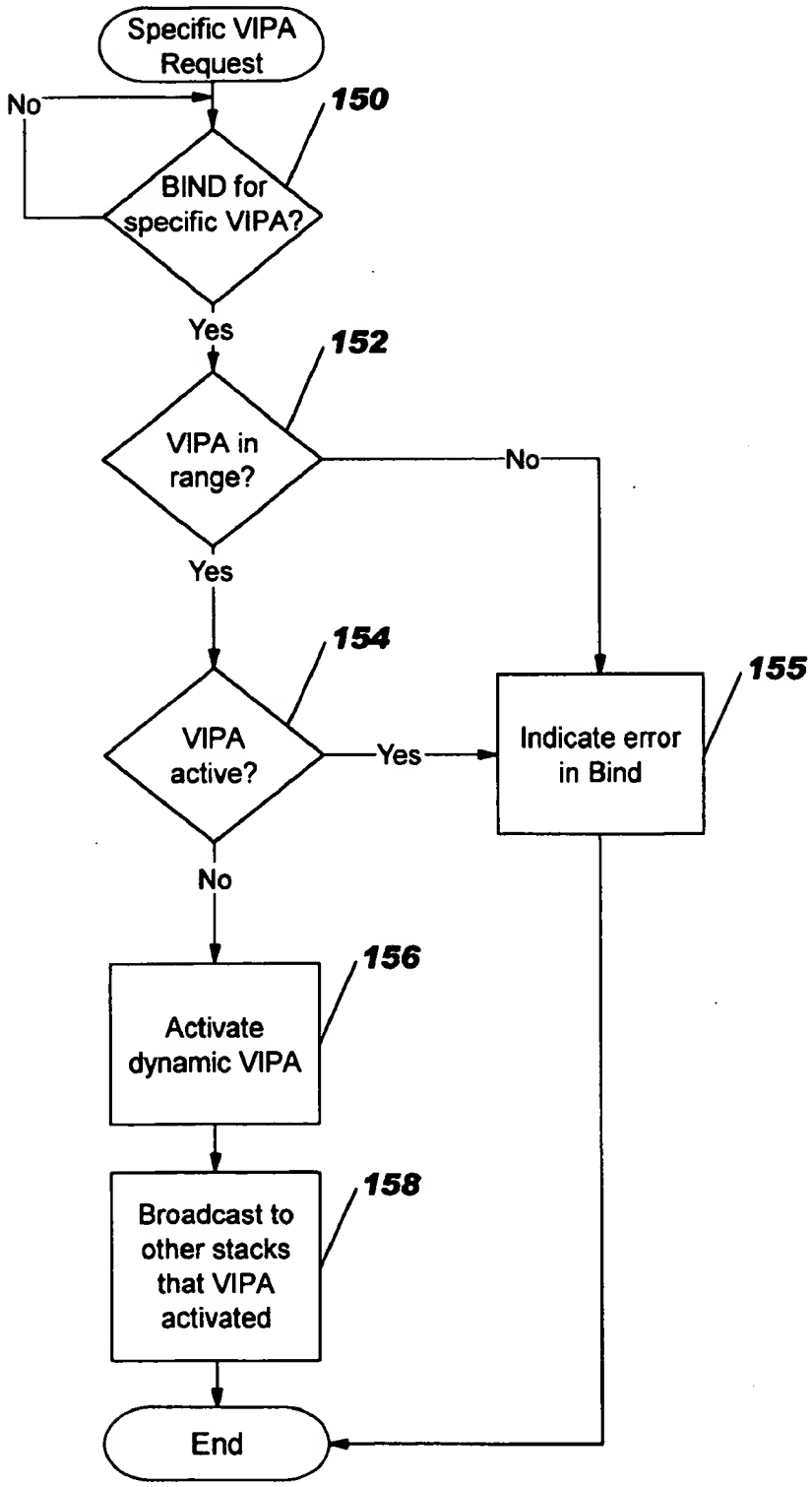

Specific VIPA Request
No
150
BIND for specific VIPA?
Yes
152
VIPA in range?
No
Yes
154
VIPA active?
Yes
155
Indicate error in Bind
No
156
Activate dynamic VIPA
158
Broadcast to other stacks that VIPA activated
End

FIG. 6

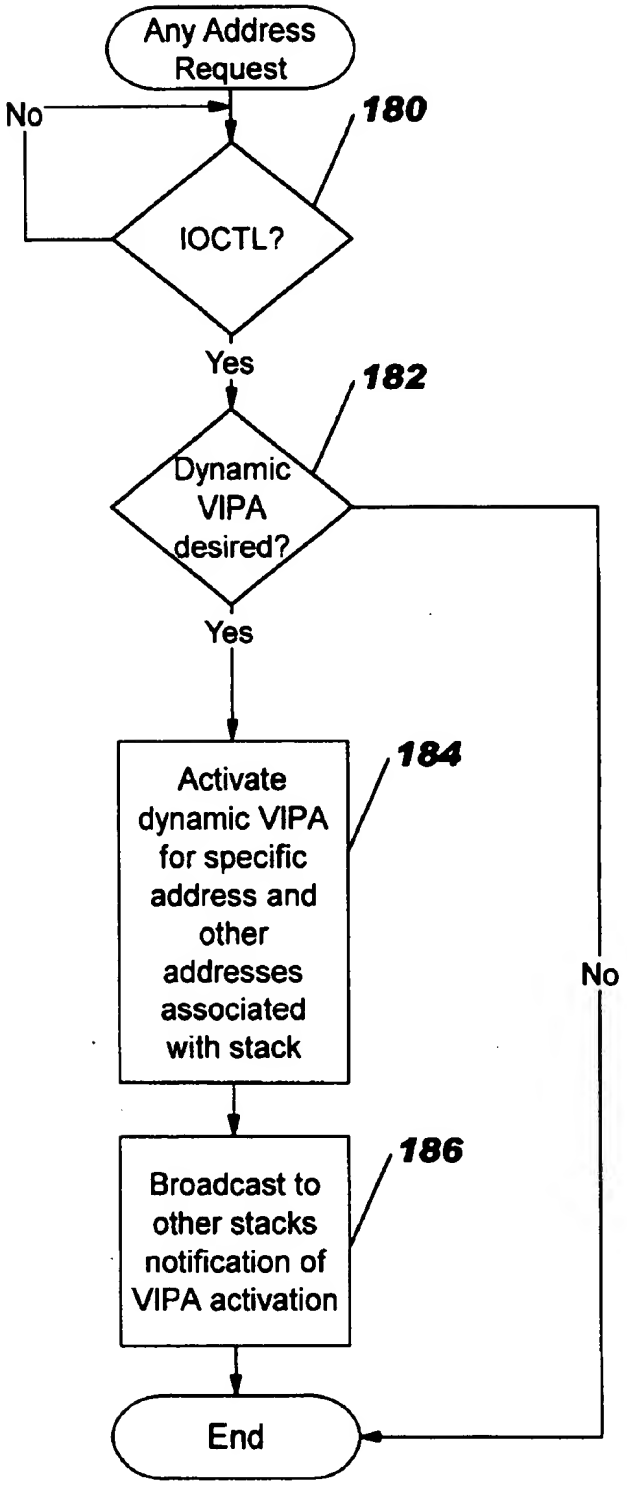

Any Address Request
No
180
IOCTL?
Yes
182
Dynamic VIPA desired?
Yes
No
184
Activate dynamic VIPA for specific address and other addresses associated with stack
186
Broadcast to other stacks notification of VIPA activation
End

FIG. 7

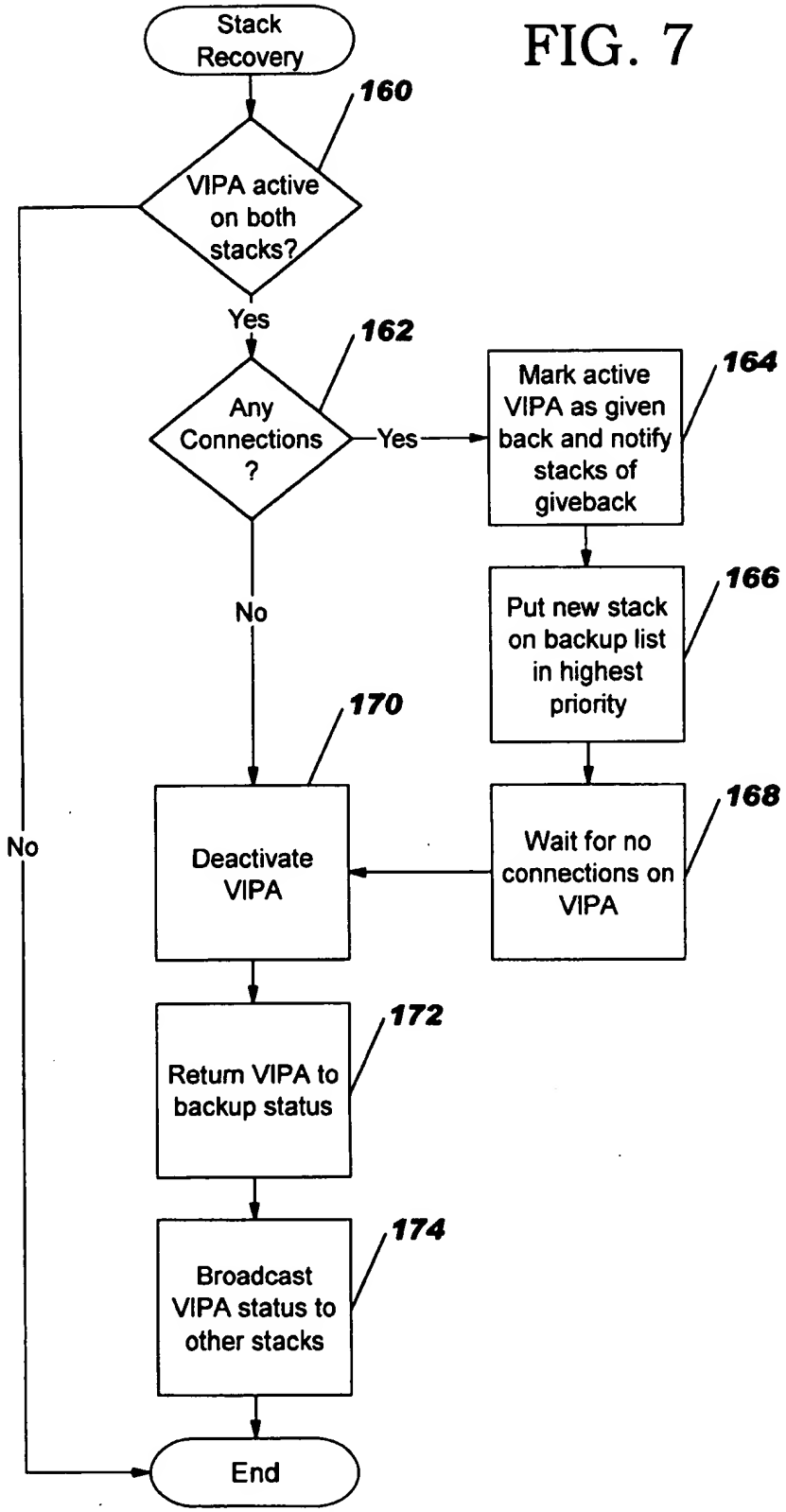

Stack Recovery
160
VIPA active on both stacks?
Yes
No
162
Any Connections ?
Yes
No
164
Mark active VIPA as given back and notify stacks of giveback
166
Put new stack on backup list in highest priority
168
Wait for no connections on VIPA
170
Deactivate VIPA
172
Return VIPA to backup status
174
Broadcast VIPA status to other stacks
End

FIG. 8

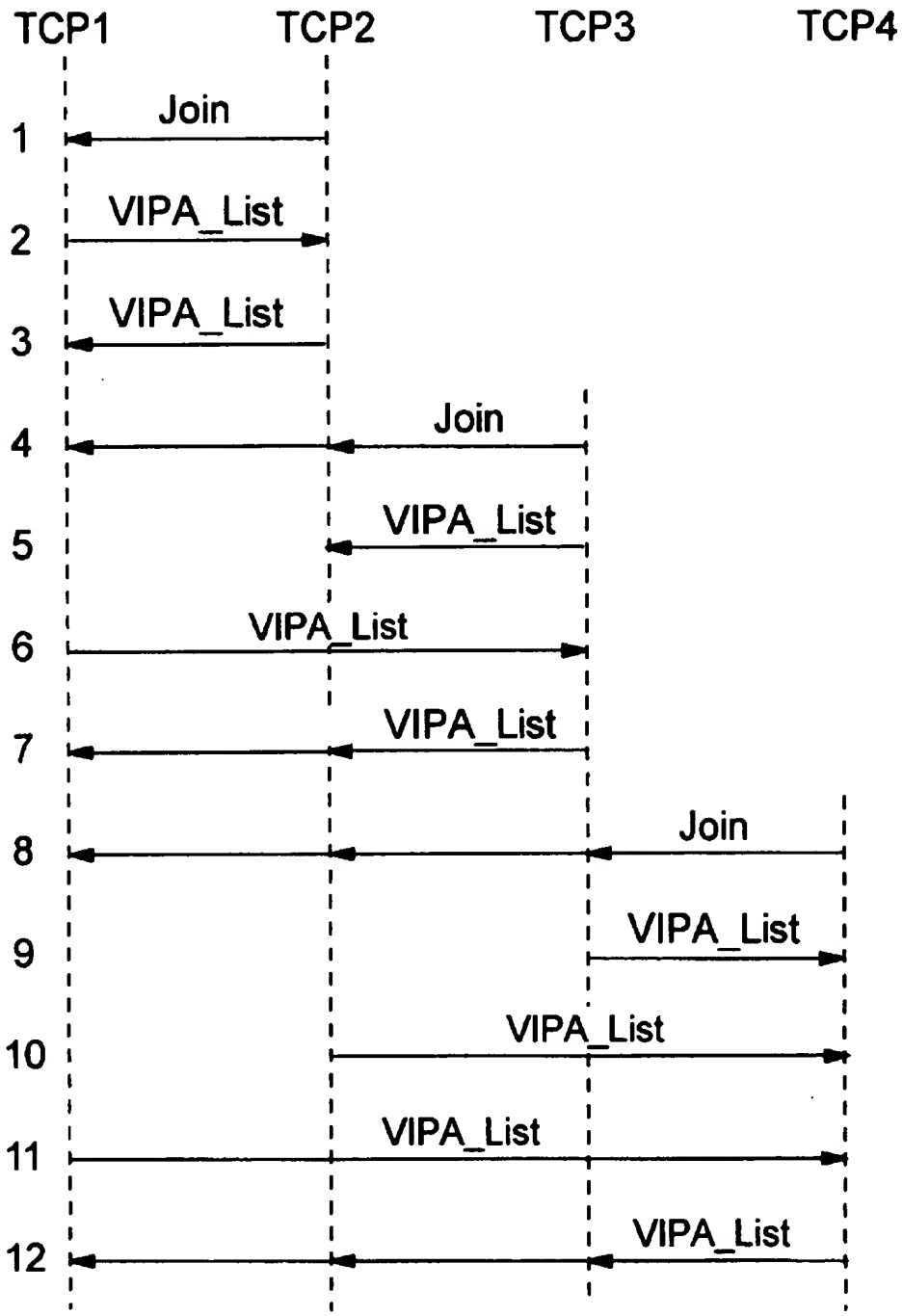

TCP1
TCP2
TCP3
TCP4
1
Join
2
VIPA_List
3
VIPA_List
4
Join
5
VIPA_List
6
VIPA_List
7
VIPA_List
8
Join
9
VIPA_List
10
VIPA_List
11
VIPA_List
12
VIPA_List

FIG. 9

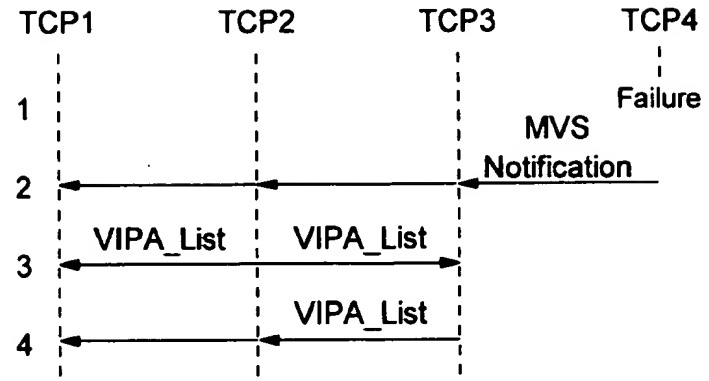
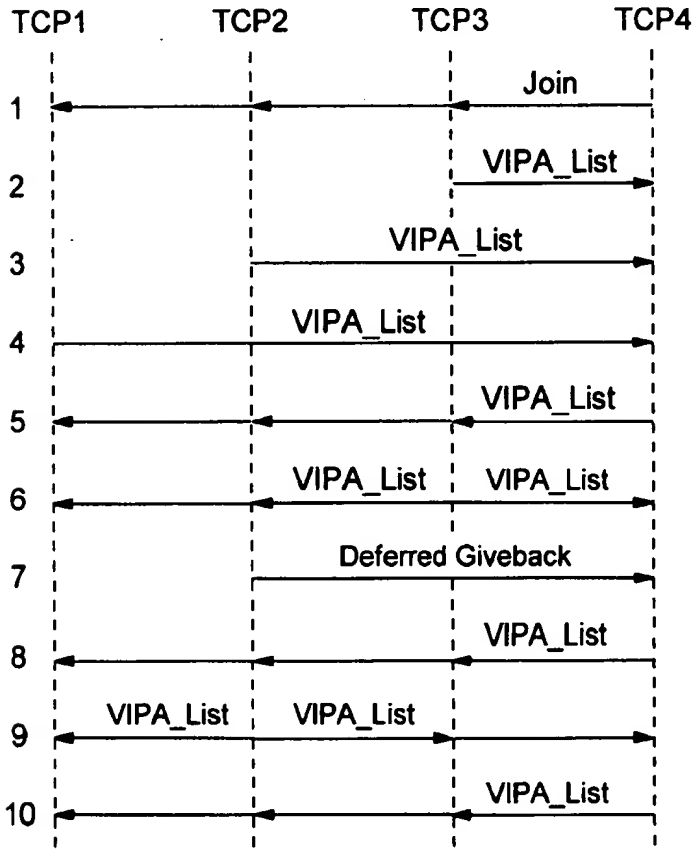

TCP1
TCP2
TCP3
TCP4
Failure
1
MVS
Notification
2
VIPA_List
VIPA_List
3
VIPA_List
4

FIG. 10


TCP1
TCP2
TCP3
TCP4
Join
1
VIPA_List
2
VIPA_List
3
VIPA_List
4
VIPA_List
5
VIPA_List
VIPA_List
6
Deferred Giveback
7
VIPA_List
8
VIPA_List
VIPA_List
9
VIPA_List
10

METHODS, SYSTEMS AND COMPUTER PROGRAM PRODUCTS FOR AUTOMATED MOVEMENT OF IP ADDRESSES WITHIN A CLUSTER

FIELD OF THE INVENTION

The present invention relates to network communications and more particularly to network communications to a cluster of data processing systems.

BACKGROUND OF THE INVENTION

The Internet Protocol (IP) is a connectionless protocol. IP packets are routed from originator through a network of routers to the destination. All physical adapter devices in such a network, including those for client and server hosts, are identified by an IP Address which is unique within the network. One valuable feature of IP is that a failure of an intermediate router node or adapter will not prevent a packet from moving from source to destination, as long as there is an alternate path through the network.

In Transmission Control Protocol/Internet Protocol (TCP/IP), TCP sets up a connection between two endpoints, identified by the respective IP addresses and a port number on each. Unlike failures of an adapter in an intermediate node, if one of the endpoint adapters (or the link leading to it) fails, all connections through that adapter fail, and must be reestablished. If the failure is on a client workstation host, only the relatively few client connections are disrupted, and usually only one person is inconvenienced. However, an adapter failure on a server means that hundreds or thousands of connections may be disrupted. On a S/390 with large capacity, the number may run to tens of thousands.

To alleviate this situation, International Business Machines Corporation introduced the concept of a Virtual IP Address, or VIPA, on its TCP/IP for OS/390 V2R5 (and added to V2R4 as well). A VIPA is configured the same as a normal IP address for a physical adapter, except that it is not associated with any particular device. To an attached router, the TCP stack on OS/390 simply looks like another router. When the TCP stack receives a packet destined for one of its VIPAs, the inbound IP function of the TCP stack notes that the IP address of the packet is in the TCP stack's Home list of IP addresses and forwards the packet up the TCP stack. The "home list" of a TCP stack is the list of IP addresses which are "owned" by the TCP stack. Assuming the TCP stack has multiple adapters or paths to it (including a Cross Coupling Facility (XCF) path from other TCP stacks in a Sysplex), if a particular physical adapter fails, the attached routing network will route VIPA-targeted packets to the TCP stack via an alternate route. The VIPA may, thus, be thought of as an address to the stack, and not to any particular adapter.

While the use of VIPAs may remove hardware and associated transmission media as a single point of failure for large numbers of connections, the connectivity of a server can still be lost through a failure of a single stack or an MVS image. The VIPA Configuration manual for OS/390 tells the customer how to configure the VIPA(s) for a failed stack on another stack, but this is a manual process. Substantial down time of a failed MVS image or TCP stack may still result until operator intervention to manually reconfigure the TCP stacks in a Sysplex to route around the failed TCP stack or MVS image.

While merely restarting an application with a new IP address may resolve many failures, applications use IP addresses in different ways and, therefore, such a solution

may be inappropriate. The first time a client resolves a name in its local domain, the local Dynamic Name Server (DNS) will query back through the DNS hierarchy to get to the authoritative server. For a Sysplex, the authoritative server should be DNS/Workload Manager (WLM). DNS/WLM will consider relative workloads among the nodes supporting the requested application, and will return the IP address for the most appropriate available server. IP addresses for servers that are not available will not be returned. The Time to Live of the returned IP address will be zero, so that the next resolution query (on failure of the original server, for example) will go all the way back to the DNS/WLM that has the knowledge to return the IP address of an available server.

However, in practice, things do not always work as described above. For example, some clients are configured to a specific IP address, thus requiring human intervention to go to another server. However, the person using the client may not have the knowledge to reconfigure the client for a new IP address. Additionally, some clients ignore the Time to Live, and cache the IP address as long as the client is active. Human intervention may again be required to recycle the client to obtain a new IP address. Also, DNSs are often deployed as a hierarchy to reduce network traffic, and DNSs may cache the IP address beyond the stated Time to Live even when the client behaves quite correctly. Thus, even if the client requests a new IP address, the client may receive the cached address from the DNS. Finally, some users may prefer to configure DNS/WLM to send a Time to Live that is greater than zero, in an attempt to limit network-wide traffic to resolve names. Problems arising from these various scenarios may be reduced if the IP address with which the client communicates does not change. However, as described above, to affect such a movement of VIPAs between TCP stacks requires operator intervention and may result in lengthy down times for the applications associated with the VIPA.

Previous approaches to increased availability focused on providing spare hardware. The High-Availability Coupled Multi-Processor (HACMP) design allows for taking over the MAC address of a failing adapter on a shared medium (LAN). This works both for a failing adapter (failover to a spare adapter on the same node) or for a failing node (failover to another node via spare adapter or adapters on the takeover node.) Spare adapters are not used for IP traffic, but they are used to exchange heartbeats among cluster nodes for failure detection. All of the work on a failing node goes to a single surviving node. In addition to spare adapters, and of course access to the same application data, the designated failover node must also have sufficient spare processing capacity to handle the entire failing node workload with "acceptable" service characteristics (response and throughput).

Automatic restart of failing applications also provides faster recovery of a failing application or node. This may be acceptable when the application can be restarted in place, but is less useful when the application is moved to another node, unless the IP address known to the clients can be moved with the application, or dynamic DNS updates with alternate IP addresses can be propagated to a DNS local to clients sufficiently quickly.

Other attempts at error recovery have included the EDDIE system described in a paper titled "EDDIE, A Robust and Scalable Internet Server" by A. Dahlin, M. Froberg, J. Grebeno, J. Walerud, and P. Winroth, of Ericsson Telecom AB, Stockholm, Sweden, May 1998. In the EDDIE approach a distributed application called "IP Address Migration Application" controls all IP addresses in the cluster. The

cluster is connected via a shared-medium LAN. IP address aliasing is used to provide addresses to individual applications over a single adapter, and these aliases are located via Address Resolution Protocol (ARP) and ARP caches in the TCP/IPs. The application monitors all server applications and hardware, and reallocates aliased IP addresses in the event of failure to surviving adapters and nodes. This approach allows applications of a failing node to be distributed among surviving nodes, but it may require the monitoring application to have complete knowledge of the application and network adapter topology in the cluster. In this sense, it is similar to existing Systems Management applications such as those provided by International Business Machines Corporation's Tivoli® network management software, but the IP Address Migration Application has direct access to adapters and ARP caches. The application also requires a dedicated IP address for inter-application communication and coordination.

In light of the above discussion, a need exists for improvements in recovery or movement of IP addresses within a cluster of data processing systems.

SUMMARY OF THE INVENTION

In view of the above discussion, it is an object of the present invention to provide for automatic recovery from failures of a communication protocol stack in a cluster of computers.

A further object of the present invention is to allow for recovery from protocol stack failures without requiring clients or servers connected to the failed protocol stacks to update cached addresses associated with the failed stack.

Still another object of the present invention is to allow automatic recovery from a protocol stack failure without requiring dedicated backup hardware.

Another object of the present invention is to provide for recovery from a protocol stack failure without reservation of capacity on a recovering system.

Yet another object of the present invention is to allow automatic recovery from a protocol stack failure without requiring additional monitoring applications or additional addresses dedicated to a recovery mechanism.

A still further object of the present invention is to allow automatic recovery from a protocol stack failure without requiring a single backup to be capable of handling the entire workload of the failed protocol stack.

These and other objects of the present invention may be provided by methods, systems, and computer program products which provide dynamic Virtual IP Addresses (VIPAs). Dynamic VIPAs are VIPAs which may be automatically moved from one protocol stack to another and are, thus, associated with an application or application group (recoverable entity) as opposed to a network adapter or host/system.

Thus, the present invention may provide for transferring a Virtual IP Address (VIPA) from a first application instance to a second application instance, where the first application instance and the second application instance are executing on a cluster of data processing systems having a plurality of communication protocol stacks associated therewith and where the first application instance is associated with a first of the plurality of communication protocol stacks and the second application instance is associated with a second of the plurality of communication protocol stacks by distributing among the plurality of communication protocol stacks a list of dynamic VIPAs. A hierarchy of backup communication protocol stacks for the dynamic VIPAs is determined based on the list of dynamic VIPAs. Upon receiving notification of failure of the first communication protocol stack the second communication protocol stack evaluates the hierarchy of backup communication protocol stacks to determine if it is the next communication protocol stack in the hierarchy of backup communication protocol stacks for the VIPA associated with the first application instance. If so, then the VIPA associated with the first application instance is transferred to the second communication protocol stack associated with the second application instance.

In a further embodiment of the present invention where a specific address is associated with an application instance, transfer of such specific dynamic VIPAs may be accomplished by establishing a range of dynamic VIPAs associated with the communication protocol stacks which are valid specific dynamic VIPAs. Information as to the bind status of dynamic VIPAs associated with the communication protocol stacks is then distributed among the communication protocol stacks to allow the communication protocol stacks to determine if a dynamic VIPA has been bound to an application instance. Upon notification of failure of a communication protocol stack the bind status of dynamic VIPAs bound to the failed communication protocol stack is reset so as to allow bind calls to the dynamic VIPAs bound to the failed communication protocol stack. When a request to bind the second instance of the application with the first VIPA of the first application instance is received at another communication protocol stack, the second application instance is bound to the first VIPA at the second communication protocol stack so as to allow communications to the second application instance through the second communication protocol stack utilizing the first VIPA.

By communicating between the protocol stacks, the hierarchy of backups may be established so that the protocol stacks themselves may handle takeover of dynamic VIPAs for failed stacks or operating system images. Furthermore, this recovery may be automatic as the protocol stacks are notified of the failure of another protocol stack in the cluster and has available sufficient information to know which protocol stack should take over a dynamic VIPA of a failed protocol stack. Thus, the automatic recovery may be accomplished without the need for a global monitoring application to control the transfer of addresses as the transfer is controlled by the protocol stacks themselves. Also, because the recovery utilizes VIPAs, clients or servers connected to the failed protocol stacks may not need to update cached addresses associated with the failed stack. Because the dynamic VIPAs associated with a failed stack are transferred to other protocol stacks in the cluster, there may be no need to provide dedicated backup hardware.

In particular embodiments of the present invention, the list of dynamic VIPAs is distributed by broadcasting from communication protocol stacks supporting dynamic VIPAs, a VIPA define message which provides identification of active dynamic VIPAs associated with the broadcasting communication protocol stack and which provides identification of dynamic VIPAs for which the broadcasting communication protocol stack is a backup communication protocol stack. Furthermore, the VIPA define message may include a priority associated with the dynamic VIPAs for which the broadcasting communication protocol stack is a backup communication protocol stack.

When a communication protocol stack receives a broadcast VIPA define message, the communication protocol stack may establish, for each VIPA in the VIPA define message, a list of communication protocol stacks associated

with the VIPA so as to provide a list of the hierarchy of backup communication protocol stacks associated with the VIPA. Preferably, the list of communications protocol stacks is in the order of priority of the communication protocol stacks.

In a still further embodiment of the present invention, if a received broadcast VIPA define message defines a communication protocol stack as a primary communication protocol stack for a dynamic VIPA identified as an active dynamic VIPA at the communication protocol stack receiving the VIPA define message, then the active dynamic VIPA may be deleted at the receiving communication protocol stack. Furthermore, the deletion of the active dynamic VIPA may be delayed if an active connection exists to the active dynamic VIPA. If so, then a VIPA delayed giveback message may be sent so as to notify other communication protocol stacks that the communication protocol stack which received the VIPA define message will delete the active dynamic VIPA when there are no connections to the active dynamic VIPA. Also, the communication protocol stack which transmitted the VIPA define message, responsive to receiving the VIPA delayed giveback message, may then broadcast a VIPA define message identifying the communication protocol stack which transmitted the VIPA define message as a backup protocol stack for the active dynamic VIPA having a highest priority.

In yet another embodiment of the present invention, notification of the restart of the first communication protocol stack may be provided to other of the plurality of communication protocol stacks. If the first communication protocol stack is the primary communication protocol stack associated with an active dynamic VIPA of the second communication protocol stack, then the active dynamic VIPA is deleted at the second communication protocol stack. The second communication protocol stack, responsive to deleting the active dynamic VIPA, may then broadcast a VIPA define message identifying the second communication protocol stack as a backup protocol stack for the active dynamic VIPA. As described above, the deletion of the active dynamic VIPA may be delayed if a connection exists to the active dynamic VIPA.

In a particular preferred embodiment of the present invention, dynamic VIPAs in the range of dynamic VIPAs and dynamic VIPAs in the list of dynamic VIPAs are mutually exclusive.

In another embodiment of the present invention, the request to bind the second application to the first VIPA at the second of the plurality of communication protocol stacks is a BIND call which specifies the first VIPA.

Furthermore, where a request is made for a specific dynamic VIPA, it may be determined if the requested VIPA is active on a communication protocol stack other than the communication protocol stack receiving the request. If so, the bind request may be rejected. If the bind call is not rejected, however, the communication protocol stacks may be notified that the requested dynamic VIPA is active at a communication protocol stack.

In yet another embodiment of the present invention, the request to bind an application to a specific dynamic VIPA is an IOCTL command which specifies the specific dynamic VIPA. In such a case, the dynamic VIPA may be activate on the communication protocol stack irrespective of whether the dynamic VIPA is active another communication protocol stack. The dynamic VIPA may then be deleted from all communication protocol stacks other than the communication protocol stack on which it was activated.

As will further be appreciated by those of skill in the art, the present invention may be embodied as methods, apparatus/systems and/or computer program products.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is block diagram of a cluster of data processing systems incorporating the present invention;

FIG. 2 is a flowchart illustrating operations for initialization of a communication protocol stack incorporating VIPA takeover according to the present invention;

FIG. 3 is a flowchart illustrating operations of a communication protocol stack incorporating VIPA takeover according to the present invention upon receiving a VIPA takeover message from another communication protocol stack;

FIG. 4 is a flowchart illustrating operations for recovery from a communication protocol stack failure for a communication protocol stack incorporating VIPA takeover according to the present invention;

FIG. 5 is a flowchart illustrating operations of a communication protocol stack incorporating VIPA takeover according to the present invention upon receiving a request for a specific dynamic VIPA address;

FIG. 6 is a flowchart illustrating operations of a communication protocol stack incorporating VIPA takeover according to the present invention associated with a specific dynamic VIPA address;

FIG. 7 is a flowchart illustrating operations of a communication protocol stack incorporating VIPA takeover according to the present invention upon recovery of a failed communication protocol stack;

FIG. 8 is a flow diagram illustrating an example communication sequence for initialization of communication protocol stacks incorporating VIPA takeover according to the present invention;

FIG. 9 is a flow diagram illustrating an example communication sequence upon failure of a communication protocol stack incorporating VIPA takeover according to the present invention; and

FIG. 10 is a flow diagram illustrating an example communication sequence upon recover of a failed communication protocol stack incorporating VIPA takeover according to the present invention.

DETAILED DESCRIPTION OF THE INVENTION

The present invention now will be described more fully hereinafter with reference to the accompanying drawings, in which preferred embodiments of the invention are shown. This invention may, however, be embodied in many different forms and should not be construed as limited to the embodiments set forth herein; rather, these embodiments are provided so that this disclosure will be thorough and complete, and will fully convey the scope of the invention to those skilled in the art. Like numbers refer to like elements throughout.

As will be appreciated by those of skill in the art, the present invention can take the form of an entirely hardware embodiment, an entirely software (including firmware, resident software, micro-code, etc.) embodiment, or an embodiment containing both software and hardware aspects. Furthermore, the present invention can take the form of a computer program product on a computer-usable or computer-readable storage medium having computer-usable or computer-readable program code means embodied in the

medium for use by or in connection with an instruction execution system. In the context of this document, a computer-usable or computer-readable medium can be any means that can contain, store, communicate, propagate, or transport the program for use by or in connection with the instruction execution system, apparatus, or device.

The computer-usable or computer-readable medium can be, for example, but is not limited to, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, device, or propagation medium. More specific examples (a nonexhaustive list) of the computer-readable medium would include the following: an electrical connection having one or more wires, a removable computer diskette, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), an optical fiber, and a portable compact disc read-only memory (CD-ROM). Note that the computer-usable or computer-readable medium could even be paper or another suitable medium upon which the program is printed, as the program can be electronically captured, via, for instance, optical scanning of the paper or other medium, then compiled, interpreted, or otherwise processed in a suitable manner if necessary, and then stored in a computer memory.

The present invention can be embodied as systems, methods, or computer program products which allow for other communication protocol stacks in a cluster of data processing systems to automatically take over Virtual IP Addresses (VIPAs) upon failure of another communication protocol stack in the cluster. VIPAs capable of such automatic takeover are referred to herein as "dynamic VIPAs." While the present invention is described with reference to a specific embodiment in an OS/390 Sysplex, as will be appreciated by those of skill in the art, the present invention may be utilized in other systems where clusters of computers utilize virtual addresses by associating an application or application group rather than a particular communications adapter with the addresses. Thus, the present invention should not be construed as limited to the particular exemplary embodiment described herein.

A cluster of data processing systems is illustrated in FIG. **1** as a cluster nodes in Sysplex **10**. As seen in FIG. **1**, several data processing systems **20**, **24**, **28**, **32** and **36** are interconnected in a Sysplex **10**. The data processing systems **20**, **24**, **28**, **32** and **36** illustrated in FIG. **1** may be operating system images, such as MVS images, executing on one or more computer systems. While the present invention will be described primarily with respect to the MVS operating system executing in an OS/390 environment, the data processing systems **20**, **24**, **28**, **32** and **36** may be mainframe computers, mid-range computers, servers or other systems capable of supporting Virtual IP Addresses and which are capable of error recovery as described herein.

As is further illustrated in FIG. **1**, the data processing systems **20**, **24**, **28**, **32** and **36** have associated with them communication protocol stacks **22**, **26**, **30**, **34** and **38**, which may be TCP/IP stacks. The communication protocol stacks **22**, **26**, **30**, **34** and **38** have been modified to incorporate a VIPA takeover function **23** as described herein for automatically transferring dynamic VIPAs from a failing communication protocol stack to a function communication protocol stack.

While each of the communication protocol stacks **22**, **26**, **30**, **34** and **38** illustrated in FIG. **1** incorporate the VIPA takeover function **23**, not all communication protocol stacks in a Sysplex need incorporate the VIPA takeover function **23**. Thus, the present invention may be carried out on any system where two or more communication protocol stacks in a cluster of data processing systems support VIPA takeover. If a communication protocol stack does not support VIPA takeover, then the VIPA takeover messages according to the present invention would be ignored by the communication protocol stack. Thus, the present invention provides backward compatibility with existing communication protocol stacks.

As is further seen in FIG. **1**, the communication protocol stacks **22**, **26**, **30**, **34** and **38** may communicate with each other through a coupling facility **40** of the Sysplex **10**, for example, utilizing XCF messaging. Furthermore, the communication protocol stacks **22**, **26**, **30**, **34** and **38** may communicate with an external network **44** such as the Internet, an intranet, a Local Area Network (LAN) or Wide Area Network (WAN) utilizing the Enterprise System Connectivity (ESCON) **42**. Thus, a client **46** may utilize network **44** to communicate with an application executing on an MVS image in Sysplex **10** through the communication protocol stacks **22**, **26**, **30**, **34** and **38**.

As is further illustrated in FIG. **1**, as an example of utilization of the present invention and for illustration purposes, data processing system **20** has associated with it communication protocol stack **22** which is associated with MVS image MVS **1** which has application APP A executing on MVS image MVS **1** and utilizing communication protocol stack **22** to allow access to, for example, client **46** through network **44**. Similarly, data processing system **24** has associated with it communication protocol stack **26** which is associated with MVS image MVS **2** which has a second instance of application APP A and an instance of application APP B executing on MVS image MVS **2** which may utilize communication protocol stack **26** for communications. Data processing system **28** has associated with it communication protocol stack **30** which is associated with MVS image MVS **3** which has a second instance of application APP B executing on MVS image MVS **3** which may utilize communication protocol stack **30** for communications. Data processing system **32** has associated with it communication protocol stack **34** which is associated with MVS image MVS **4** which has a third instance of application APP A executing on MVS image MVS **4** which may utilize communication protocol stack **34** for communications. Finally, data processing system **36** has associated with it communication protocol stack **38** which is associated with MVS image MVS **5** which has a third instance of application APP B executing on MVS image MVS **5** which may utilize communication protocol stack **38** for communications.

Utilizing the above described system configuration as an example, the VIPA takeover function **23** will now be described. The VIPA takeover function **23** automates the movement of the dynamic VIPA to an appropriate surviving communication protocol stack in the event of failure. One aspect of the VIPA takeover function **23** according to the present invention allows automated movement to a stack where an existing suitable application instance already resides, allowing that instance to serve clients formerly going to the failed stack/node. Another aspect of the VIPA takeover function **23** allows dynamic VIPAs to be defined and activated through application action without modifying the applications themselves so that the dynamic VIPA moves when the application is moved. Thus, application instances on a failing cluster node may be distributed among many (or all) of the surviving nodes, allowing for reducing the amount of spare processing capacity that must be provisioned for each node that may participate in takeover. Where a dynamic

VIPA is used simply to gain access to one of many suitable application instances, movement of that dynamic VIPA on stack or node failure is automatically handled by surviving stacks. Where a dynamic VIPA is tied to a specific application instance, the dynamic VIPA will be activated wherever the application instance is started—or moved.

For completely automated takeover, the communication protocol stacks **22, 26, 30, 34** and **38** may be configured as to which stack is primary, and which stacks may serve as backup—and in what preferred order. Different dynamic VIPAs may have different sets of backup stacks, possibly overlapping. For dynamic VIPAs tied to application instances, configuration will limit the specific values of the IP addresses of the dynamic VIPAs that may be started, so that a keying error in configuring a dynamic VIPA will not attempt to start an IP address that is not available to the installation or the organization. Within the constraints defined by customer configuration of the TCP/IPs in the cluster (Sysplex), however, there is complete freedom in initial placement and movement of application instances.

Server application instances, such as APP A and APP B of FIG. **1**, may be deployed across the Sysplex **10** in different ways. In some cases, appropriate application instances are already available on multiple Sysplex nodes, serving any IP address. Dynamic VIPAs may be used to direct traffic to specific stacks or MVS images based on load, but other existing server instances would do equally well. However, it is preferred that a moved dynamic VIPA end up on a stack serving one of the existing application instances.

In other cases, the dynamic VIPA is specific to an application instance which may have unique data or state. In this case, if the application instance is restarted elsewhere, the dynamic VIPA should move along with it to a stack on the same MVS image. Note that these application instances may either bind to a specific IP address (the dynamic VIPA) or to any IP address; the requirement to move the dynamic VIPA to the right stack is the same in both cases.

Thus, the VIPA takeover function **23** of the communication protocol stacks **22, 26, 30, 34** and **38** preferably provides for three VIPA takeover scenarios. In particular, the VIPA takeover function **23** preferably provides for VIPA takeover where multiple homogeneous application instances are already executing utilizing differing communication protocol stacks. In such a case the communication protocol stacks **22, 26, 30, 34** and **38** collaborate to move the dynamic VIPA of a failing communication protocol stack to an appropriate communication protocol stack(as configured by the Sysplex administrator). Also, if an application instance binds to a specific (dynamic VIPA) address, the communication protocol stacks **22, 26, 30, 34** and **38** ensure that only one stack "owns" that dynamic VIPA at any one time. Finally, the VIPA takeover function **23** preferably allows an application instance to bind to any IP address (INADDR_ANY), but also allows association a specific dynamic VIPA with that specific application instance.

Turning now to the first of these scenarios, for the present example, application APP A has dynamic VIPAs VA1–VA3 assigned to the respective first, second and third instances of APP A; and application APP B likewise has VIPAs VB1–VB3 assigned to the respective first, second and third instances of APP B.

Configuration of a takeover hierarchy may be provided by a new definition block established by a system administrator for each communication protocol stack **22, 26, 30, 34** and **38**. The new definition block defines dynamic VIPAs for which a communication protocol stack operates as the primary communication protocol stack and defines dynamic VIPAs for which the communication protocol stack acts as a backup communication protocol stack. Thus, a definition block "VIPADynamic" may be defined as

```
VIPADynamic
Dynamic VIPA definitions . . .
ENDVIPADynamic
```

The definitions within the VIPADynamic block for primary ownership are: ti VIPADEFine address_mask VIPA1 VIPA2 . . .

where the address_mask is used to determine the network prefix to be advertised to routing daemons for OSPF or RIP. Both network prefix (sometimes known as subnet address) and the mask will be advertised to the routing daemon. All of the VIPAs in a single VIPADEFine statement must belong to the same subnet, network, or supernet, as determined by the network class and address mask.

The definitions within the VIPADynamic block for backup are:

```
VIPABackup rank VIPAx VIPAy . . .
```

where the rank is a number between 1 and 254 used to determine relative order within the backup chain(s) for the associated dynamic VIPA(s). A communication protocol stack with the higher rank will take over the dynamic VIPAs before a communication protocol stack with a lower rank.

In this first scenario, the communication protocol stacks **22, 26, 30, 34** and **38** collaborate entirely among themselves to move dynamic VIPAs when a communication protocol stack or the hosting MVS fails. As part of the initialization process of the communication protocol stacks **22, 26, 30, 34** and **38**, a takeover hierarchy is defined through messages broadcast by the VIPA takeover function **23** of each of the communication protocol stacks **22, 26, 30, 34** and **38** to the other communication protocol stacks **22, 26, 30, 34** and **38**. At initialization or profile changes, the communication protocol stacks **22, 26, 30, 34** and **38** communicate to all partner communication protocol stacks the complete list of dynamic VIPAs, both primary and backup defined for the communication protocol stack.

When a communication protocol stack **22, 26, 30, 34** and **38** or MVS fails, the respective MVS for each surviving communication protocol stack notifies the communication protocol stack of the failure. The first stack (highest rank) in the backup list for each dynamic VIPA then creates and activates the VIPA dynamically and automatically, and advertises the dynamic VIPA to associated routing daemons for dynamic routing updates in attached routers. It is up to the system/cluster or DNS administrator to ensure that there is in fact a suitable application instance already in place and ready to serve new connection requests to the dynamic VIPA.

When the failed communication protocol stack is restarted, its primary dynamic VIPAs will only be activated when there are no active TCP connections to that dynamic VIPA on other communication protocol stacks. If other communication protocol stacks have active connections, activation on the 'primary' communication protocol stack will be delayed.

Returning to the example of FIG. **1**, for MVS1 to MVS5 the VIPADEFine statements may be:

```
MVS1: VIPADEFine VA1
MVS2: VIPADEFine VA2 VB1
MVS3: VIPADEFine VB2
MVS4: VIPADEFine VA3
MVS5: VIPADEFine VB3
```

For purposes of illustration the respective address masks have been omitted because they are only significant to the routing daemons.

The order of the stacks in the backup lists for each dynamic VIPA is determined by the relative rank values on each VIPABackup statement. For clarity of the present example, the example was constructed such that each node needed only one VIPABackup statement with all backup dynamic VIPAs in it. A more complex example could have resulted in the need for a different rank value on each individual dynamic VIPA on each stack. This is handled by having individual VIPABackup statements with fewer dynamic VIPAs, or in the ultimate case only one dynamic VIPA for each. Note also that the rank value for a dynamic VIPA is significant only for other stacks specifying that same dynamic VIPA. In the example shown, both MVS1 and MVS3 have a rank value of **30**. There is NOT a tie, because the dynamic VIPAs specified in the VIPABackup statement on MVS1 are different from the dynamic VIPAs specified on MVS3.

Thus, the VIPABackup statements for MVS1 to MVS5 of FIG. **1** may be:

MVS1: VIPABackup **30** VA2 VA3

MVS2: VIPABackup **20** VA1 VA3 VB2 VB3

MVS3: VIPABackup **30** VB1 VB3

MVS4: VIPABackup **40** VA1 VA2

MVS5: VIPABackup **10** VB1 VB2

For purposes of illustration, assume that MVS2 fails, taking with it the communication protocol stack **26** and the instances of applications APP A and APP B. MVS1, MVS3, MVS4 and MVS5 notify their respective communication protocol stacks **22**, **30**, **34** and **38** of the failure. For VA2, the dynamic VIPA for the instance of application APP A on MVS2, the first communication protocol stack in the backup chain is the communication protocol stack **34** on MVS4, since it was defined to have the highest VIPABackup rank (**40**) for VA2. MVS4, therefore, defines and activates dynamic VIPA VA4 in addition to VA3 to allow access to an instance of application APP A. Similarly, the communication protocol stack **30** on MVS3 defines and activates dynamic VIPA VB1 in addition to VB2 to allow access to an instance of application APP B.

At some time later, MVS2 and the communication protocol stack **26** and application instances for APP A and APP B are restarted. Assume further that at that time, application APP A does not have connections to VA4 on MVS4, but application APP B has connections via VB1 on MVS3. When the communication protocol stack **26** is started on MVS2, it notifies the other stacks of its intent to define and activate VA2 and VB1. The communication protocol stack **34** on MVS4 already has VA2 active, but does not have active connections through VA2, so it deletes VA2 and reverts to backup status. The communication protocol stack **26** on MVS2 thus "owns" VA4, and can advertise it to the routers via dynamic routing protocols.

On the other hand, the communication protocol stack **30** on MVS3 not only has VB2 active, but it also has active connections through VB1. It notifies the communication protocol stack **26** on MVS2 of this fact, and sets a flag that another stack desires VB1 when there are no more connections. The communication protocol stack **26** on MVS2 deactivates and deletes VB1, and declares itself backup for VB1 with a rank of **255**. This puts the stack at the head of the backup chain, since the highest rank that can be configured is **254**.

When the last connection to VB1 on the communication protocol stack **30** on MVS3 is closed, the stack deletes VB1, declares itself backup for VB1 at the former rank of **40**, and notifies the other stacks. The communication protocol stack **26** on MVS2 notes that VB3 has transitioned from active to inactive, and that it is the first stack in the backup chain, so it defines and activates VB1. The Sysplex has now reverted to the stable pre-failure configuration.

As can be seen from the above example, when the original failed stack is restarted, giveback is graceful, in the sense that connections established on the backup communication protocol stack are not disrupted when the failing stack comes back up. If the installation wants traffic to revert to the restarted stack immediately, regardless of active connections, the system operator on the backup stack can issue VARY OBEY on a profile with VIPADELETE and VIPABACKUP for that dynamic VIPA. The VIPADELETE will delete the dynamic VIPA along with any active connections, and the VIPABackup will re-establish the communication protocol stack as a backup stack.

With other types of IP addressing, including conventional VIPAs, the stack reports the IP addresses in the Home list to DNS/WLM when the stack is activated. This means that DNS/WLM has an accurate picture of active IP addresses for each stack. However, DNS/WLM will report such IP addresses in resolving any supported application on that stack. To avoid having connections established to applications other than the ones associated with dynamic VIRAs, no Dynamic VIPAs will be reported by the stack to DNS/WLM. Dynamic VIPAs should be configured statically in DNS and associated with the corresponding applications.

Furthermore, this first scenario does not require any additional configuration for the application, since the application instances if the applications accept connection requests to any IP address. However, the Sysplex Administrator should ensure that Sysplex communication protocol stacks are configured properly, and that application instances remain active on all active stacks that may be backup for dynamic VIPAs associated with the application. The communication protocol stacks may have no knowledge of application instances at all. Thus, for example, in restarting MVS2, Applications App A and App B should be restarted prior to restarting the communication protocol stack **26**, to ensure that they are ready to accept connection requests as soon as the dynamic VIPAs are reactivated on the communication protocol stack **26** on MVS2.

In the second scenario described above the application instance is configured to bind to a specific IP address and each application instance is distinct from other application instances. In other words, the first instance of APP is not the same as the second instance of APP A, even though both represent instances of application APP A. The difference may be specific requests that may be satisfied, or different backing data bases, or whatever other differences may exist. In this example, application instances are deployed as they were for the first scenario, but the respective dynamic VIPAs need to move with the application instances.

Another difference from the prior scenario is that failure of the application instance, stack, or MVS means that the application instance will be restarted elsewhere in the Sysplex. How this restart is accomplished is not determined by VIPA Takeover function **23** but could be by operator intervention, by Automatic Restart Manager (ARM), or other Sysplex-wide mechanism.

As described above, in this second scenario the application instance is configured to bind to a specific IP address. Normally, such a BIND( ) is successful only if the specified

IP address is active on the communication protocol stack to which the application issues the BIND( ). The VIPA Takeover function **23** enhances the communication protocol stacks **22, 26, 30, 34** and **38** such that if the IP address is not in use on another communication protocol stack, and the communication protocol stack has been so configured, a dynamic VIPA will be activated on the target communication protocol stack. "In use" as used herein refers to active connections to that IP address on a communication protocol stack.

Because the communication protocol stacks **22, 26, 30, 34** and **38** exchange information about active IP addresses (including Dynamic VIPAs), the determination of whether a dynamic VIPA is in use can be made fairly quickly. If the dynamic VIPA is in use on another communication protocol stack, or the target communication protocol stack has not been configured to accept this particular request, the BIND( ) will fail with ENOADDR EADDRNOTAVAILBLE as it would for communication protocol stacks without the VIPA Takeover function **23**.

The dynamic VIPAs to be activated in this manner should be grouped into subnets of their own. As an example, the VIPAs for instances of Application APP A could be grouped into one subnet with network prefix (subnet address) SN1 and address mask SNAM1, and similarly for Application APP B. A subnet could be defined for all applications, or a different subnet for each application, or combinations thereof. For the present example, application APP A dynamic VIPAs are assumed to be in SN1, and dynamic VIPAs for instances of Application APP B in SN2 with address mask SNAM2.

A configuration option to designate for each communication protocol stack **22, 26, 30, 34** and **38** the allowable subnet or subnets in which dynamic VIPAs may be activated dynamically by BIND(specific address) is also provided. The VIPARange statement does this:

VIPARANGE address_mask network_prefix where network_prefix is any IP address within the subnet, and address_mask is used to determine the subnet address to be advertised to the communication protocol stack's routing daemon. As many VIPARange statements may be specified on a communication protocol stack as needed to designate all subnets in which dynamic VIPAs may be activated. The subnets in VIPARange should not normally be the same as the subnets for VIPADEFine/VIPABackup, since a Dynamic VIPA may not simultaneously participate in automatic takeover (VIPADEFineNIPABackup) and be activated via BIND( ).

Once activated on a communication protocol stack via BIND( ), a dynamic VIPA remains active until the activating application terminates (address automatically deactivated by TCP/IP) unless the dynamic VIPA is moved to another stack. The system operator may delete an active dynamic VIPA by issuing VARY OBEY with a profile containing a VIPADELETE statement referencing the specific dynamic VIPA. When the VIPADELETE statement is processed, the dynamic VIPA is deactivated and deleted from the communication protocol stack. If the dynamic VIPA was activated by a BIND( ), it will not be activated automatically on another communication protocol stack until an application issues a BIND( ) to the same address at the other communication protocol stack.

For this second scenario, instances of application APP A may be deployed on MVS1, MVS2, and MVS4. Since the corresponding VIPAs have been organized into subnet SN1 with address mask SNAM1 the corresponding communication protocol stacks **22, 26** and **34** are configured with the following statement:

```
VIPARange SNAM1 SN1
```

Similarly, communication protocol stacks **26, 30** and **38** on MVS2, MVS3 and MVS5 have the following statement in a VIPADynamic block:

```
VIPARange SNAM2 SN2
```

When the properly configured application instances are started on the respective MVS images, the BIND( ) in each instance will cause activation of the corresponding dynamic VIPA. It does not matter which application instance is started on which stack/MVS image. Indeed, all of the instances of an application could be started on the same MVS image. Alternatively, there could be more stacks with the application VIPARange statement than actual started application instances. However, an appropriate VIPARange must be configured on a stack before an application can successfully BIND to a dynamic VIPA on that stack.

Assume now that MVS2 fails, similar to the prior scenario. The other communication protocol stacks **22, 30, 34** and **38** are notified of the failure of the MVS2, and they take note of the fact that dynamic VIPAs VA2 and VB1 are no longer active in the Sysplex **10**. When the failure is detected, the second instance of application instance APP A is restarted (by ARM or a human) on MVS4. MVS4 has already been configured with the appropriate VIPARange for application APP A, so the BIND(VA2) by the second application instance APP A causes VA2 to be defined and activated on the communication protocol stack **34** on MVS4. Similarly, when the first instance of application APP B moves to MVS3, dynamic VIPA VB1 is activated on the associated communication protocol stack **30**.

If, however, only the second instance of application APP A fails at MVS2, the communication protocol stack **26** notes the failure and clears the associated connections. These dynamic VIPAs are not advertised to DNS/WLM as general stack addresses, so it is reasonable to assume that there are no more connections to VA2 on MVS2. Thus, if the second instance of the application APP A is restarted on MVS1 (or MVS4), the BIND(VA2) causes deletion of VA2 on the communication protocol stack **26** on MVS2, and definition and activation of VA2 on MVS1 (or MVS4).

If the entire MVS2 fails and is restarted, along with the communication protocol stack **26** and the second instance of application APP A and the first instance of application APP B, MVS and communication protocol stack activation will proceed to completion. However, the BIND to VA2 or VB1 will succeed only if an application or TCP/IP previously bound to the dynamic VIPA is no longer running. If these instances have been restarted on other stacks while MVS2 and its communication protocol stack **26** are being restarted, the instances must be quiesced (via application mechanisms) or stopped before restarting those same instances on MVS2.

Application restart happens entirely independent of any stack function. Other than freeing up active connections on catastrophic application failure, the stack is not really aware of the application per se. This means that the restart manager may be as sophisticated as desired in determining on which stack or MVS image to restart a failed application instance, including consulting WLM. The solution works equally well for operator-managed restart or Network Management automation of the movement.

The final scenario is very similar to the second scenario, except that the application instance does not bind to a particular IP address. The Sysplex Administrator nonetheless desires to associate a particular IP address (dynamic VIPA) with each specific application instance. Otherwise, the application scenarios are identical.

If the application instance desires to BIND( ) to INADDR_ANY to accept connection requests to any IP address supported by the communication protocol stack, there is no implicit indication to the stack of which dynamic VIPA should be activated. However, an IOCTL command may be defined which allows the issuer to command the stack to activate a dynamic VIPA. This IOCTL may be made public, so new applications can exploit it if desired. Activation of the dynamic VIPA will succeed as long as the designated IP address is not claimed by one of the other communication protocol stacks as an IP address for a physical interface or a static VIPA, or defined via VIPADEFine or VIPABackup. The issuer must be APF Authorized or have Root Authority. Given this level of authority, if the Dynamic VIPA was activated earlier on another stack via BIND( ) or IOCTL, the dynamic VIPA will be deleted on the other stack, even if this means that existing connections are broken.

Communication protocol stack configuration to allow for this third scenario is the same as for the second scenario, namely a VIPARange defining a subnet containing the designated VIPA. There is no configuration difference between activation by BIND( ) or IOCTL, and the same VIPARange may be used for both if desired. Since the same dynamic VIPA may not be activated by IOCTL/BIND while also participating in automatic takeover as defined by VIPADEFine/VIPABackup, it is preferred that subnets for VIPADEFine be different from subnets for VIPARange.

This new IOCTL command may be used by new applications that desire to BIND(INADDR ANY) while still being associated with a specific dynamic VIPA. The application configuration should specify the dynamic VIPA or VIPAs to be used with IOCTL at application instance initialization.

To address the case where the application cannot be modified, possibly because the application has been purchased from a vendor and the source code is not available, a utility may be utilized that may be initiated via JCL, from the OMVS command line, or from a shell script. The utility may support a parameter specifying the dynamic VIPA to be activated. The utility may also be used to delete a dynamic VIPA, as an alternative to VARY OBEY by a system operator. However, the utility must be APF authorized or be executed from a user ID with Root authority. The utility may return an indication of success or failure which may be tested.

Preferably, each application instance utilizing this third scenario should include configuration information either specifying the dynamic VIPA for use by the IOCTL command or an additional JCL step to invoke the utility to activate the dynamic VIPA. In the event of failure, the recovery mechanism may be the same as that of the second scenario described above.

Having described the present invention with reference to FIG. **1**, more detailed operations of the present invention will now be described with reference to FIGS. **2** through **7** which are flowchart illustrations of operations according to embodiments of the present invention. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer program instructions. These program instructions may be provided to a processor to produce a machine, such that the instructions which execute on the processor create means for implementing the functions specified in the flowchart and/or block diagram block or blocks. The computer program instructions may be executed by a processor to cause a series of operational steps to be performed by the processor to produce a computer implemented process such that the instructions which execute on the processor provide steps for implementing the functions specified in the flowchart and/or block diagram block or blocks.

Accordingly, blocks of the flowchart illustrations and/or block diagrams support combinations of means for performing the specified functions, combinations of steps for performing the specified functions and program instruction means for performing the specified functions. It will also be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by special purpose hardware-based systems which perform the specified functions or steps, or combinations of special purpose hardware and computer instructions.

FIG. **2** illustrates operations carried out by one embodiment of the VIPA takeover function **23** during initialization. As seen in FIG. **2**, a dynamic VIPA range is established for the communication protocol stack (block **100**). As described above, this range may be established through a VIPARange definition in the configuration block of the communication protocol stack. The primary and backup dynamic VIPAs are also established for the communication protocol stack (block **102**). The primary and backup dynamic VIPAs may be established, as described above, utilizing the VIPADefine and VIPABackup definitions in the configuration block for the communication protocol stack. After establishing the primary and backup configuration, this information is broadcast to the other communication protocol stacks in the Sysplex **10** utilizing XCF messaging (block **104**).

FIG. **3** illustrates operations of the VIPA takeover function **23** when the VIPA takeover function **23** receives the XCF messages for initialization or modification of the VIPA takeover function **23**. As seen in FIG. **3**, the VIPA takeover function **23** waits for messages from other communication protocol stacks (block **110**). When a message is received, it is determined if the message contains information for activation via bind or IOCTL for a dynamic VIPA within a range for a communication protocol stack (block **112**) and if so, then the dynamic VIPA is associated with the source communication protocol stack of the message and that information maintained by the receiving communication protocol stack (block **114**).

It is also determined if the message contains primary or backup information about a communication protocol stack (block **116**). If so, then it is determined if the primary/backup information causes contention with a dynamic VIPA of the receiving communication protocol stack (block **118**). Contention may occur, for example, if the message states that the source communication protocol stack is the primary communication protocol stack for a dynamic VIPA which the receiving communication protocol stack also "owns" as primary.

If there is no contention with the primary/backup definitions then the source communication protocol stack is identified as the primary stack for the dynamic VIPAs defined as primary in the message and the source communication protocol stack is placed at the top of the backup hierarchy for those dynamic VIPAs (block **120**). The backup hierarchy may be a list in backup order of the communication protocol stacks associated with a dynamic VIPA. The source communication protocol stack is also identified as the backup stack for the dynamic VIPAs defined as backup in the message and the source communication protocol stack is placed in the backup hierarchy for those dynamic VIPAs based on the priority assigned to the backup (block **122**).

Thus, for example, the source communication protocol stack would be placed higher in the list a dynamic VIPA than communication protocols with either lower priority or with the same priority but whose message was received earlier. Thus, contention of backups with the same priority may be resolved by utilizing the most recently received message as the tie breaker for communication protocols with the same priority.

Once the hierarchy for each dynamic VIPA is revised based on the received message, the communication protocol stack determines if there are any dynamic VIPAs which do not have a primary identified and for which the communication protocol stack is the highest backup stack. If so, then the VIPA takeover function **23** activates these dynamic VIPAs on the communication protocol stack (block **123**). Thus, a dynamic VIPA may be transferred to a backup stack by the primary stack issuing a message which removes the primary stack as the primary stack for the dynamic VIPA. To activate the dynamic VIPA, the communication protocol stack creates device, link and home statements for the communication protocol stack specifying the dynamic VIPA. The communication protocol stack also notifies the routing daemon through either a BSDROUTINGPARMS entry for ORouteD routing daemons or by sending notification to the routing daemon listening socket for OMPROUTE routing daemons.

Returning to block **118**, if contention for a primary dynamic VIPA is detected, then it is determined if the conflicting VIPA is in the home list of another communication protocol stack (block **124**).The maintenance of a home list by each communication protocol stack and broadcasting this home list to other communication protocol stacks may be handled utilizing conventional communication protocol stack communications and, therefore, need not be incorporated in the VIPA takeover function **23**. If so, then the communication protocol stack receiving the message deletes the conflicting dynamic VIPA from its dynamic VIPA defines (block **126**) and broadcasts this change in VIPA definitions to the other communication protocol stacks (block **128**). The maintenance of a home list by each communication protocol stack and broadcasting this home list to other communication protocol stacks may be handled utilizing conventional communication protocol stack communications and, therefore, need not be incorporated in the VIPA takeover function **23**. However, the VIPA takeover function **23** should have access to the home list information for each communication protocol stack in the Sysplex. If there are more conflicting dynamic VIPAs, then the conflict resolution process is begun again (block **118**), otherwise, the operations continue with block **120** as described above.

If the conflicting dynamic VIPA is not in the home list of another communication protocol stack, then it is determined if there are active connections on the conflicting dynamic VIPA on the communication protocol stack receiving the message (block **130**). If there are active connections, then the restart recovery operations described with reference to FIG. **7** below are carried out (block **134**). If there are more conflicting dynamic VIPAs, then the conflict resolution process is begun again (block **118**), otherwise, the operations continue with block **120** as described above.

If there are not connections on the conflicting dynamic VIPA (block **130**), then the dynamic VIPA is given back to the communication protocol stack from which the message was received (block **132**) and the receiving communication protocol stack reverts to its backup status for the dynamic VIPA and updates its backup hierarchy accordingly. This new status of the receiving communication protocol stack is then broadcast to the other stacks (block **128**). If there are more conflicting dynamic VIPAs, then the conflict resolution process is begun again (block **118**), otherwise, the operations continue with block **120** as described above.

Returning to block **117**, if the message does not contain primary or backup definitions, then it is determined if the message is a delayed giveback message (block **117**). As described above, a delayed giveback message would be sent by a communication protocol stack which was going to giveback a dynamic VIPA to a primary communication protocol stack but had to wait for connections to the dynamic VIPA to terminate before the giveback could occur. If the message is a delayed giveback message, then the receiving communication protocol revises it dynamic VIPA definitions to place the dynamic VIPA of the giveback message as a backup with a priority of **255** and broadcasts these new definitions to the other communication protocol stacks (block **119**).

FIG. **4** illustrates operations of a VIPA takeover function **23** when notified of a communication protocol stack failure by MVS. As seen in FIG. **4**, when a stack failure is detected (block **130**), the communication protocol stacks reset the dynamic VIPAs in the range of dynamic VIPAs of the failed communication protocol stack to indicate that the dynamic VIPAs are inactive and, therefore, a BIND( ) to those dynamic VIPAs will be allowed (block **132**). The communication protocol stack then identifies the dynamic VIPAs for which the failed communication protocol stack was the primary communication protocol stack (block **134**). If the communication protocol stack is not the highest priority backup communication protocol stack for any of the identified dynamic VIPAs (block **136**), then no further action need be taken. However, if the communication protocol stack is the highest priority backup communication protocol stack, then the communication protocol stack activates those dynamic VIPAs (block **138**). The communication protocol stack then updates its dynamic VIPA definitions to reflect its new status a primary and broadcasts this information to the other communication protocol stacks (block **140**).

FIG. **5** illustrates operations carried out by a communication protocol stack when it receives a BIND( ) to a specific dynamic VIPA. When ta BIND( ) to a specific VIPA is received (block **150**) it is determined if the requested dynamic VIPA is within the range of dynamic VIPAs specified for the communication protocol stack (block **152**). If the requested VIPA is not in the range, then the BIND( ) fails and an error is indicated (block **155**). If the requested VIPA is within the range, then it is determined if the dynamic VIPA is active on any other communication protocol stacks (block **154**). If so, then an error is also indicated and the BIND( ) fails (block **155**). However, if the requested VIPA is within the range of dynamic VIPAs and is not active on another communication protocol stack, then the dynamic VIPA is activated (block **156**) and this activation broadcast to the other communication protocol stacks (block **158**).

FIG. **6** illustrates the operations of the present invention which an application BINDs to any IP address but where an operator, JCL or other mechanism also utilizes the IOCTL command described above to activate a dynamic VIPA associated with the application. As seen in FIG. **6**, when an IOCTL is received (block **180**) it is determined if a specific dynamic VIPA is to be associated with the application (block **182**). If so, then the dynamic VIPA is activated (block **184**) and this activation broadcast to the other communication protocol stacks (block **186**). Note that the activation of the dynamic VIPA in the present case occurs irrespective of whether the dynamic VIPA is active on other communication

protocol stacks. If the dynamic VIPA is active on other communication protocol stacks, the dynamic VIPA will be deleted from the other communication protocol stacks as described above.

FIG. 7 illustrates operations of the VIPA takeover function 23 when a failed communication protocol stack is recovered. As briefly described above with reference to FIG. 3, the operations of FIG. 7 may take place as a result of contention between definitions of primary communication protocol stacks for a dynamic VIPA. Thus, as seen at block 160, if a dynamic VIPA is active at two communication protocol stacks, then it is determined if there are any active connections to the dynamic VIPA at the communication protocol stack which received the message indicating that another communication protocol stack was the primary communication protocol stack for a dynamic VIPA (block 162). If there are active connections, then the dynamic VIPA id marked as given back and the other communication protocol stacks notified of this delayed giveback status (block 164). The new primary communication protocol stack is also placed on the backup list at the highest priority as a result of the new communication protocol stack broadcasting a new backup definition responsive to receiving the delayed giveback message (block 166). The communication protocol stack which is giving back the dynamic VIPA then waits until there are no more connections to the dynamic VIPA (block 168).

When there are no connections to the dynamic VIPA (block 168), or if there originally were no connections to the dynamic VIPA (block 162), the dynamic VIPA is deactivated (block 170) and returned to backup status (block 172). The new status of the dynamic VIPA for the communication protocol stack is then broadcast to the other communication protocol stacks (block 174). As a result, when the communication protocol stack which originally broadcast the message which resulted in the contention for the dynamic VIPA receives the message that the dynamic VIPA has been returned to backup status, because this originating communication protocol stack is the highest priority backup stack, it will assume primary ownership of the dynamic VIPA and then broadcast a message to reflect this change in status (see e.g. FIG. 3, block 123).

FIGS. 8 through 10 are flow diagrams illustrating example communications between communication protocol stacks utilizing the present invention. FIG. 8 is a flow diagram of the communications for initialization between four TCP stacks (TCP1 through TCP4) incorporating VIPA takeover function 23. As seen in FIG. 8, TCP1 is the first initialized TCP stack. Thus, at time 1, the following example hierarchy for dynamic VIPAs V1 through V8 may exist:

V1
TCP1, Time-1, Active
V2
TCP1, Time-1, Active
V3
TCP1, Time-1, Backup (10)
V4
V5
TCP1, Time-1, Backup (10)
V6
V7
TCP1, Time-1, Backup (10)
V8
TCP1, Time-1, Backup (10)

At time 1, TCP2 joins the TCP XCF group of TCP stacks and TCP1 is notified of the join. In response to the Join notification, at time 2, TCP1 transmits its VIPA definitions (VIPA_List) to TCP2, which, also transmits its VIPA_List to TCP 1 at time 3. As a result, the VIPA hierarchy in TCP1 and TCP2 will be:

V1
TCP1, Time-1, Active
TCP2, Time-2, Backup (20)
V2
TCP1, Time-1, Active
TCP2, Time-2, Backup (20)
V3
TCP2, Time-2, Active
TCP1, Time-1, Backup (10)
V4
TCP2, Time-2, Active
V5
TCP1, Time 1, Backup (10)
V6
TCP2, Time-2, Backup (20)
V7
TCP2, Time-2, Backup (20)
TCP1, Time-1, Backup (10)
V8
TCP1, Time-1, Backup (10)

At time 4, TCP3 has been initialized sufficiently to join the TCP XCF group and, therefore, TCP2 and TCP1 are notified of the join. In response to the Join, TCP2 at time 5 sends its VIPA_List to TCP3 and TCP1 at time 6 sends its VIPA_List to TCP3. In response, TCP3 broadcasts its VIPA_List to TCP1 and TCP2. As a result, the VIPA hierarchy in TCP1, TCP2 and TCP3 will be:

V1
TCP1, Time-1, Active
TCP2, Time-2, Backup (20)
V2
TCP1, Time-1, Active
TCP3, Time-3, Backup (30)
TCP2, Time-2, Backup (20)
V3
TCP2, Time-2, Active
TCP3, Time-3, Backup (30)
TCP1, Time-1, Backup (10)
V4
TCP2, Time-2, Active
TCP3, Time-3, Backup (30)
V5
TCP3, Time-3, Active
TCP1, Time 1, Backup (10)
V6
TCP3, Time-3, Active
TCP2, Time-2, Backup (20)
V7
TCP2, Time-2, Backup (20)
TCP1, Time-1, Backup (10)
V8
TCP3, Time-3, Backup (30)
TCP1, Time-1, Backup (10)

At time 8, TCP4 has been initialized sufficiently to join the TCP XCF group and, therefore, TCP3, TCP2 and TCP1 are notified of the join. In response to the Join, TCP3 at time 9 sends its VIPA_List to TCP4, TCP2 at time 10 sends its

VIPA_List to TCP4 and TCP1 at time 11 sends its VIPA_List to TCP4. In response, TCP4 broadcasts its VIPA_List to TCP1, TCP2 and TCP3. As a result, the VIPA hierarchy in TCP1, TCP2, TCP3 and TCP4 will be:

V1
TCP1, Time-1, Active
TCP2, Time-2, Backup (20)
V2
TCP1, Time-1, Active
TCP3, Time-3, Backup (30)
TCP2, Time-2, Backup (20)
V3
TCP2, Time-2, Active
TCP3, Time-3, Backup (30)
TCP1, Time-1, Backup (10)
V4
TCP2, Time-2, Active
TCP3, Time-3, Backup (30)
V5
TCP3, Time-3, Active
TCP1, Time 1, Backup (10)
V6
TCP3, Time-3, Active
TCP2, Time-2, Backup (20)
V7
TCP4, Time-4, Active
TCP2, Time-2, Backup (20)
TCP1, Time-1, Backup (10)
V8
TCP4, Time-4, Active
TCP3, Time-3, Backup (30)
TCP1, Time-1, Backup (10)

Thus, after distributing each TCP stack's VIPA_List to the other TCP stacks, each stack should have the same backup hierarchy.

FIG. 9 illustrates to the communication flows when a TCP stack, such as TCP4, fails. As seen in FIG. 9, after time 1, TCP4 fails. At time 2, MVS notifies TCP1, TCP2 and TCP3 of the failure of failure. Each of the remaining TCP stacks remove the dynamic VIPAs associated with TCP4 from the hierarchy. Because TCP1 is not the highest priority backup for the active dynamic VIPAs V7 and V8 on TCP4, after removing references to TCP4 from the hierarchy TCP1 takes no further action until new VIPA_Lists are received from TCP2 and TCP3. Because TCP2 is the highest priority backup for V7, TCP2 activates V7 and at time 3 sends a revised VIPA_List reflecting this activation to TCP1 and TCP3. TCP3 also determines that it is the highest priority backup for dynamic VIPA V8 and, therefore, activates V8 and at time 4 sends a revised VIPA_List to TCP1 and TCP2. As a result of TCP4's failure, after the takeover of the VIPAs for which TCP4 was active has occurred, the hierarchy will be as follows:

V1
TCP1, Time-1, Active
TCP2, Time-2, Backup (20)
V2
TCP1, Time-1, Active
TCP3, Time-3, Backup (30)
TCP2, Time-2, Backup (20)
V3
TCP2, Time-2, Active
TCP3, Time-3, Backup (30)
TCP1, Time-1, Backup (10)
V4
TCP2, Time-2, Active
TCP3, Time-3, Backup (30)
V5
TCP3, Time-3, Active
TCP1, Time 1, Backup (10)
V6
TCP3, Time-3, Active
TCP2, Time-2, Backup (20)
V7
TCP2, Time-5, Active
TCP1, Time-1, Backup (10)
V8
TCP3, Time-6, Active
TCP1, Time-1, Backup (10)

Thus as reflected in the hierarchy, TCP3 will be active for V8 and TCP2 will be active for V7.

FIG. 10 illustrates to the communication flows when a TCP stack which has previously failed, such as TCP4, is recovered. As seen in FIG. 10, at time 1, TCP4 is recovered sufficiently to join the XCF group and TCP1, TCP2 and TCP3 are notified of this join. Thus, at time 2 TCP3 sends its VIPA_List to TCP4, at time 3, TCP2 sends its VIPA_List to TCP4 and at time 4, TCP1 sends its VIPA_List to TCP4. At time 5, TCP4 broadcasts it VIPA_List to TCP1, TCP2 and TCP3. When TCP3 receives the VIPA_List from TCP4, it determines that TCP wants to own V8 which is currently owned by TCP3. Since there are not active connections to V8 at TCP3, TCP3 deletes V8 and broadcasts, at time 6, its VIPA_List identify itself as a backup for V8.

TCP2 also evaluates the VIP_List from TCP4 and determines that TCP4 wants ownership of V7. However, because active connections exist to V7 at TCP2, TCP2, at time 7, sends a deferred giveback message to TCP4 notifying TCP4 that TCP2 will give back V7 when there are no active connections to V7. In response, at time 8, TCP4 re-broadcasts its VIPA_List identifying itself as the highest priority backup for V7. At time 9 there are no longer active connections to V7 at TCP2 so TCP2 deletes V7 and broadcasts its VIPA_List indicating that TCP2 is a backup for V7. When TCP4 received the VIPA_List from TCP2 it recognizes that there is no longer an active TCP stack for V7 and, because it is the highest priority backup, activates V7 and then re-broadcasts, at time 10, its VIPA_List reflecting the activation of V7. As a result, the hierarchy maintained at each TCP stack will revert to that before the failure of TCP4.

While the present invention has been described with respect to the VIPA takeover function as a part of the communication protocol stack, as will be appreciated by those of skill in the art, such functions may be provided as separate functions, objects or applications which may cooperate with the communication protocol stacks. Furthermore, the present invention has been described with reference to particular sequences of operations. However, as will be appreciated by those of skill in the art, other sequences may be utilized while still benefitting from the teachings of the present invention. Thus, while the present invention is described with respect to a particular division of functions or sequences of events, such divisions or sequences are merely illustrative of particular embodiments of the present invention and the present invention should not be construed as limited to such embodiments.

Furthermore, while the present invention has been described with reference to particular embodiments of the

present invention in an OS/390 environment, as will be appreciated by those of skill in the art, the present invention may be embodied in other environments and should not be construed as limited to OS/390 but may be incorporated into other systems, such as a Unix or other environments by associating applications or groups of applications with an address rather than a communications adapter. Thus, the present invention may be suitable for use in any collection of data processing systems which allow sufficient communication to all of for the use of dynamic virtual addressing. Accordingly, specific references to OS/390 systems or facilities, such as the "coupling facility," "ESCON," "Sysplex" or the like should not be construed as limiting the present invention.

In the drawings and specification, there have been disclosed typical preferred embodiments of the invention and, although specific terms are employed, they are used in a generic and descriptive sense only and not for purposes of limitation, the scope of the invention being set forth in the following claims.

What which is claimed is:

1. A method of transferring a Virtual IP Address (VIPA) from a first application instance to a second application instance, wherein the first application instance and the second application instance are executing on a cluster of data processing systems having a plurality of communication protocol stacks associated therewith and wherein the first application instance is associated with a first of the plurality of communication protocol stacks and the second application instance is associated with a second of the plurality of communication protocol stacks, the method comprising the steps of:

distributing among the plurality of communication protocol stacks a list of dynamic VIPAs, wherein a hierarchy of backup communication protocol stacks for the dynamic VIPAs in the list of dynamic VIPAs is determined based on the list of dynamic VIPAs;

receiving notification of failure of the first communication protocol stack at the second of the plurality of communication protocol stacks;

evaluating the hierarchy of backup communication protocol stacks to determine if the second of the plurality of communication protocol stacks is a next communication protocol stack of the plurality of communications after the first of the plurality of communication protocol stacks in the hierarchy of backup communication protocol stacks for the VIPA associated with the first application instance; and

transferring the VIPA associated with the first application instance to the second of the plurality of communication protocol stacks associated with the second application instance if the list of dynamic VIPAs identifies the second of the plurality of communication protocol stacks as the next communication protocol stack of the plurality of communications after the first of the plurality of communication protocol stacks in the hierarchy of backup communication protocol stacks for the VIPA associated with the first application instance.

2. A method according to claim 1, wherein the step of distributing comprises the step of broadcasting from a communication protocol stack supporting dynamic VIPAs, a VIPA define message which provides identification of active dynamic VIPAs associated with the broadcasting communication protocol stack and which provides identification of dynamic VIPAs for which the broadcasting communication protocol stack is a backup communication protocol stack.

3. A method according to claim 2, wherein the VIPA define message further comprises a priority associated with the dynamic VIPAs for which the broadcasting communication protocol stack is a backup communication protocol stack.

4. A method according to claim 3, further comprising the steps of:

receiving a broadcast VIPA define message;

establishing for each VIPA in the VIPA define message a list of communication protocol stacks associated with the VIPA so as to provide a list of the hierarchy of backup communication protocol stacks associated with the VIPA.

5. A method according to claim 4, wherein the list of communications protocol stacks is in the order of priority of the communication protocol stacks.

6. A method according to claim 4, further comprising the steps of:

determining if the received broadcast VIPA define message defines a communication protocol stack as a primary communication protocol stack for a dynamic VIPA identified as an active dynamic VIPA at the communication protocol stack receiving the VIPA define message; and

deleting the active dynamic VIPA at the receiving communication protocol stack if the received broadcast VIPA define message defines a communication protocol stack as a primary communication protocol stack for a dynamic VIPA identified as an active dynamic VIPA at the communication protocol stack receiving the VIPA define message.

7. A method according to claim 6, wherein the deleting step is preceded by the step of determining if an active connection exists to the active dynamic VIPA, and wherein the step of deleting the active dynamic VIPA is delayed if an active connection exists until there are no connections to the active dynamic VIPA.

8. A method according to claim 7, wherein the deleting step is preceded by the steps of:

broadcasting a VIPA delayed giveback message so as to notify other communication protocol stacks of the plurality of communication protocol stacks that the communication protocol stacks which received the VIPA define message will delete the active dynamic VIPA when there are no connections to the active dynamic VIPA; and

broadcasting from the communication protocol stack which transmitted the VIPA define message, responsive to receiving the VIPA delayed giveback message, a VIPA define message identifying the communication protocol stack which transmitted the VIPA define message as a backup protocol stack for the active dynamic VIPA having a highest priority.

9. A method according to claim 1, further comprising the steps of:

receiving, at the second of the plurality of communication protocol stacks, notification of the restart of the first of the plurality of communication protocol stacks;

determining if the first of the plurality of communication protocol stacks is the primary communication protocol stack associated with an active dynamic VIPA of the second of the plurality of communication protocol stacks;

deleting the active dynamic VIPA if the first of the plurality of communication protocol stacks is the primary communication protocol stack associated with the dynamic VIPA; and

broadcasting from the second of the plurality of communication protocol stacks, responsive to deleting the

active dynamic VIPA, a VIPA define message identifying the second of the plurality of communication protocol stacks as a backup protocol stack for the active dynamic VIPA.

10. A method according to claim 9, wherein the deleting step is preceded by the step of determining if an active connection exists to the active dynamic VIPA of the second of the plurality of communication protocol stacks, and wherein the step of deleting the active dynamic VIPA is delayed if an active connection exists until there are no connections to the active dynamic VIPA.

11. A method according to claim 10, further comprising the steps of:

transmitting from the second communication protocol stack to the first of the plurality of communication protocol stacks a VIPA delayed giveback message so as to notify the first of the plurality of communication protocol stacks that the second of the plurality of communication protocol stacks will delete the active dynamic VIPA when there are no connections to the active dynamic VIPA; and

broadcasting from the first of the plurality of communication protocol stacks, responsive to receiving the VIPA delayed giveback message, a VIPA define message identifying the first of the plurality of communication protocol stacks as a backup protocol stack for the active dynamic VIPA having a highest priority.

12. A method according to claim 1, further comprising the steps of:

establishing a range of dynamic VIPAs associated with the first and the second of the plurality of communication protocol stacks;

distributing among the plurality of communication protocol stacks information as to a bind status of dynamic VIPAs associated with communication protocol stacks of the plurality of communication protocol stacks so as to allow communication protocol stacks in the plurality of communication protocol stacks to determine if a dynamic VIPA has been bound to an application instance;

receiving notification of failure of the first of the plurality of communication protocol stacks at the second of the plurality of communication protocol stacks;

resetting the bind status of dynamic VIPAs bound to the first of the plurality of communication protocol stacks so as to allow bind calls to the dynamic VIPAs bound to the first of the plurality of communication protocol stack;

receiving at the second of the plurality of communication protocol stacks a request to bind the second instance of the application with a specified dynamic VIPA of the first application instance; and

binding the second application instance to the specified dynamic VIPA at the second communication protocol stack so as to allow communications to the second application instance through the second communication protocol stack utilizing the first VIPA.

13. A method according to claim 12, wherein dynamic VIPAs in the range of dynamic VIPAs associated with communication protocol stacks of the plurality of communication protocol stacks and dynamic VIPAs in the list of dynamic VIPAs are mutually exclusive.

14. A method according to claim 12, wherein the request to bind the second application to the specified dynamic VIPA at the second of the plurality of communication protocol stacks is a BIND call which specifies the specified dynamic VIPA.

15. A method according to claim 12, further comprising the steps of:

determining if the specified dynamic VIPA is active on a communication protocol stack of the plurality of communication protocol stacks other than the second of the plurality of communication protocol stacks; and

rejecting the bind request from the second instance of the application if the specified dynamic VIPA is active on a communication protocol stack of the plurality of communication protocol stacks other than the second of the plurality of communication protocol stacks.

16. A method according to claim 12, further comprising the step of notifying the plurality of communication protocol stacks that the specified dynamic VIPA is active at the second of the plurality of communication protocol stacks.

17. A method according to claim 12, wherein the request to bind the second application to the specified dynamic VIPA at the second of the plurality of communication protocol stacks is an IOCTL command which specifies the specified dynamic VIPA.

18. A method according to claim 17, further comprising the steps of:

receiving at the second of the plurality of communication protocol stacks a BIND call specifying that the second instance of the application can bind to any valid address at the second of the plurality of communication protocol stacks; and

binding the second instance of the application to any valid address at the second of the plurality of communication protocol stacks.

19. A method according to claim 17, further comprising the steps of:

activating the specified dynamic VIPA on the second of the plurality of communication protocol stacks irrespective of whether the specified dynamic VIPA is active a communications protocol stack other than the second of the plurality of communication protocol stacks; and

deleting the specified dynamic VIPA from all communication protocol stacks other than the second of the plurality of communication protocol stacks.

20. A method of transferring a Virtual IP Address (VIPA) from a first application instance to a second application instance, wherein the first application instance and the second application instance are executing on a cluster of data processing systems having a plurality of communication protocol stacks associated therewith and wherein the first application instance is bound to a first VIPA associated with a first of the plurality of communication protocol stacks and wherein the second application instance is a restarted instance of the first application associated with a second of the plurality of communication protocol stacks, the method comprising the steps of:

establishing a range of dynamic VIPAs associated with a communication protocol stack of the plurality of communication protocol stacks;

distributing among the plurality of communication protocol stacks information as to a bind status of dynamic VIPAs associated with communication protocol stacks of the plurality of communication protocol stacks so as to allow communication protocol stacks in the plurality of communication protocol stacks to determine if a dynamic VIPA has been bound to an application instance;

receiving notification of failure of the first communication protocol stack at the second of the plurality of communication protocol stacks;

resetting the bind status of dynamic VIPAs bound to the first communication protocol stack so as to allow bind calls to the dynamic VIPAs bound to the first communication protocol stack;

receiving at the second communication protocol stack a request to bind the second instance of the application with the first VIPA of the first application instance; and

binding the second application instance to the first VIPA at the second communication protocol stack so as to allow communications to the second application instance through the second communication protocol stack utilizing the first VIPA.

21. A method according to claim **20**, wherein the request to bind the second application to the first VIPA at the second of the plurality of communication protocol stacks is a BIND call which specifies the first VIPA.

22. A method according to claim **20**, further comprising the steps of:

determining if the first VIPA is active on a communication protocol stack of the plurality of communication protocol stacks other than the second of the plurality of communication protocol stacks; and

rejecting the bind request from the second instance of the application if the first VIPA is active on a communication protocol stack of the plurality of communication protocol stacks other than the second of the plurality of communication protocol stacks.

23. A method according to claim **20**, further comprising the step of notifying the plurality of communication protocol stacks that the first VIPA is active at the second of the plurality of communication protocol stacks.

24. A method according to claim **20**, wherein the request to bind the second application to the first VIPA at the second of the plurality of communication protocol stacks is an IOCTL command which specifies the first VIPA.

25. A method according to claim **24**, further comprising the steps of:

receiving at the second of the plurality of communication protocol stacks a BIND call specifying that the second instance of the application can bind to any valid address at the second of the plurality of communication protocol stacks; and

binding the second instance of the application to any valid address at the second of the plurality of communication protocol stacks.

26. A method according to claim **24**, further comprising the steps of:

activating the first VIPA on the second of the plurality of communication protocol stacks irrespective of whether the first VIPA is active a communications protocol stack other than the second of the plurality of communication protocol stacks; and

deleting the first VIPA from all communication protocol stacks other than the second of the plurality of communication protocol stacks.

27. A system for transferring a Virtual IP Address (VIPA) from a first application instance to a second application instance, wherein the first application instance and the second application instance are executing on a cluster of data processing systems having a plurality of communication protocol stacks associated therewith and wherein the first application instance is associated with a first of the plurality of communication protocol stacks and the second application instance is associated with a second of the plurality of communication protocol stacks, comprising:

means for distributing among the plurality of communication protocol stacks a list of dynamic VIPAs, wherein a hierarchy of backup communication protocol stacks for the dynamic VIPAs in the list of dynamic VIPAs is determined based on the list of dynamic VIPAs;

means for receiving notification of failure of the first communication protocol stack at the second of the plurality of communication protocol stacks;

means for evaluating the hierarchy of backup communication protocol stacks to determine if the second of the plurality of communication protocol stacks is a next communication protocol stack of the plurality of communications after the first of the plurality of communication protocol stacks in the hierarchy of backup communication protocol stacks for the VIPA associated with the first application instance; and

means for transferring the VIPA associated with the first application instance to the second of the plurality of communication protocol stacks associated with the second application instance if the list of dynamic VIPAs identifies the second of the plurality of communication protocol stacks as the next communication protocol stack of the plurality of communications after the first of the plurality of communication protocol stacks in the hierarchy of backup communication protocol stacks for the VIPA associated with the first application instance.

28. A system according to claim **27**, further comprising:

means for establishing a range of dynamic VIPAs associated with the first and the second of the plurality of communication protocol stacks;

means for distributing among the plurality of communication protocol stacks information as to a bind status of dynamic VIPAs associated with communication protocol stacks of the plurality of communication protocol stacks so as to allow communication protocol stacks in the plurality of communication protocol stacks to determine if a dynamic VIPA has been bound to an application instance;

means for receiving notification of failure of the first of the plurality of communication protocol stacks at the second of the plurality of communication protocol stacks;

means for resetting the bind status of dynamic VIPAs bound to the first of the plurality of communication protocol stacks so as to allow bind calls to the dynamic VIPAs bound to the first of the plurality of communication protocol stack;

means for receiving at the second of the plurality of communication protocol stacks a request to bind the second instance of the application with a specified dynamic VIPA of the first application instance; and

means for binding the second application instance to the specified dynamic VIPA at the second communication protocol stack so as to allow communications to the second application instance through the second communication protocol stack utilizing the first VIPA.

29. A system for transferring a Virtual IP Address (VIPA) from a first application instance to a second application instance, wherein the first application instance and the second application instance are executing on a cluster of data processing systems having a plurality of communication protocol stacks associated therewith and wherein the first application instance is bound to a first VIPA associated with a first of the plurality of communication protocol stacks and wherein the second application instance is a restarted instance of the first application associated with a second of the plurality of communication protocol stacks, comprising:

means for establishing a range of dynamic VIPAs associated with a communication protocol stack of the plurality of communication protocol stacks;

means for distributing among the plurality of communication protocol stacks information as to a bind status of dynamic VIPAs associated with communication protocol stacks of the plurality of communication protocol stacks so as to allow communication protocol stacks in the plurality of communication protocol stacks to determine if a dynamic VIPA has been bound to an application instance;

means for receiving notification of failure of the first communication protocol stack at the second of the plurality of communication protocol stacks;

means for resetting the bind status of dynamic VIPAs bound to the first communication protocol stack so as to allow bind calls to the dynamic VIPAs bound to the first communication protocol stack;

means for receiving at the second communication protocol stack a request to bind the second instance of the application with the first VIPA of the first application instance; and

means for binding the second application instance to the first VIPA at the second communication protocol stack so as to allow communications to the second application instance through the second communication protocol stack utilizing the first VIPA.

30. A computer program product for transferring a Virtual IP Address (VIPA) from a first application instance to a second application instance, wherein the first application instance and the second application instance are executing on a cluster of data processing systems having a plurality of communication protocol stacks associated therewith and wherein the first application instance is associated with a first of the plurality of communication protocol stacks and the second application instance is associated with a second of the plurality of communication protocol stacks, comprising:

a computer readable storage medium having computer readable program code embodied in said medium, said computer readable program code comprising:

computer readable program code for distributing among the plurality of communication protocol stacks a list of dynamic VIPAs, wherein a hierarchy of backup communication protocol stacks for the dynamic VIPAs in the list of dynamic VIPAs is determined based on the list of dynamic VIPAs;

computer readable program code for receiving notification of failure of the first communication protocol stack at the second of the plurality of communication protocol stacks;

computer readable program code for evaluating the hierarchy of backup communication protocol stacks to determine if the second of the plurality of communication protocol stacks is a next communication protocol stack of the plurality of communications after the first of the plurality of communication protocol stacks in the hierarchy of backup communication protocol stacks for the VIPA associated with the first application instance; and

computer readable program code for transferring the VIPA associated with the first application instance to the second of the plurality of communication protocol stacks associated with the second application instance if the list of dynamic VIPAs identifies the second of the plurality of communication protocol stacks as the next communication protocol stack of the plurality of communications after the first of the plurality of communication protocol stacks in the hierarchy of backup communication protocol stacks for the VIPA associated with the first application instance.

31. A computer program product according to claim 30, further comprising:

computer readable program code for establishing a range of dynamic VIPAs associated with the first and the second of the plurality of communication protocol stacks;

computer readable program code for distributing among the plurality of communication protocol stacks information as to a bind status of dynamic VIPAs associated with communication protocol stacks of the plurality of communication protocol stacks so as to allow communication protocol stacks in the plurality of communication protocol stacks to determine if a dynamic VIPA has been bound to an application instance;

computer readable program code for receiving notification of failure of the first of the plurality of communication protocol stacks at the second of the plurality of communication protocol stacks;

computer readable program code for resetting the bind status of dynamic VIPAs bound to the first of the plurality of communication protocol stacks so as to allow bind calls to the dynamic VIPAs bound to the first of the plurality of communication protocol stack;

computer readable program code for receiving at the second of the plurality of communication protocol stacks a request to bind the second instance of the application with a specified dynamic VIPA of the first application instance; and

computer readable program code for binding the second application instance to the specified dynamic VIPA at the second communication protocol stack so as to allow communications to the second application instance through the second communication protocol stack utilizing the first VIPA.

32. A computer program product for transferring a Virtual IP Address (VIPA) from a first application instance to a second application instance, wherein the first application instance and the second application instance are executing on a cluster of data processing systems having a plurality of communication protocol stacks associated therewith and wherein the first application instance is bound to a first VIPA associated with a first of the plurality of communication protocol stacks and wherein the second application instance is a restarted instance of the first application associated with a second of the plurality of communication protocol stacks, comprising:

a computer readable storage medium having computer readable program code embodied in said medium, said computer readable program code comprising:

computer readable program code for establishing a range of dynamic VIPAs associated with a communication protocol stack of the plurality of communication protocol stacks;

computer readable program code for distributing among the plurality of communication protocol stacks information as to a bind status of dynamic VIPAs associated with communication protocol stacks of the plurality of communication protocol stacks so as to allow communication protocol stacks in the plurality of communication protocol stacks to determine if a dynamic VIPA has been bound to an application instance;

computer readable program code for receiving notification of failure of the first communication protocol stack at the second of the plurality of communication protocol stacks;

computer readable program code for resetting the bind status of dynamic VIPAs bound to the first communication protocol stack so as to allow bind calls to the dynamic VIPAs bound to the first communication protocol stack;

computer readable program code for receiving at the second communication protocol stack a request to bind the second instance of the application with the first VIPA of the first application instance; and

computer readable program code for binding the second application instance to the first VIPA at the second communication protocol stack so as to allow communications to the second application instance through the second communication protocol stack utilizing the first VIPA.

* * * * *

US005951650A

United States Patent [19]

Bell et al.

[11] Patent Number: 5,951,650

[45] Date of Patent: Sep. 14, 1999

[54] **SESSION TRAFFIC SPLITTING USING VIRTUAL INTERNET PROTOCOL ADDRESSES ASSOCIATED WITH DISTINCT CATEGORIES OF APPLICATION PROGRAMS IRRESPECTIVE OF DESTINATION IP ADDRESS**

[75] Inventors: **Jon Anthony Bell**, Raleigh; **Edward Glen Britton**, Chapell Hill, both of N.C.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **08/792,607**

[22] Filed: **Jan. 31, 1997**

[51] **Int. Cl.**[6] **G06F 13/00**

[52] **U.S. Cl.** **709/238**; 709/240

[58] **Field of Search** 359/136, 172; 345/502; 709/238, 240

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,731,784	3/1988	Keller et al.	359/136
5,091,849	2/1992	Davis et al.	345/502
5,321,542	6/1994	Freitas et al.	359/172
5,490,252	2/1996	Macera et al.	395/200.79

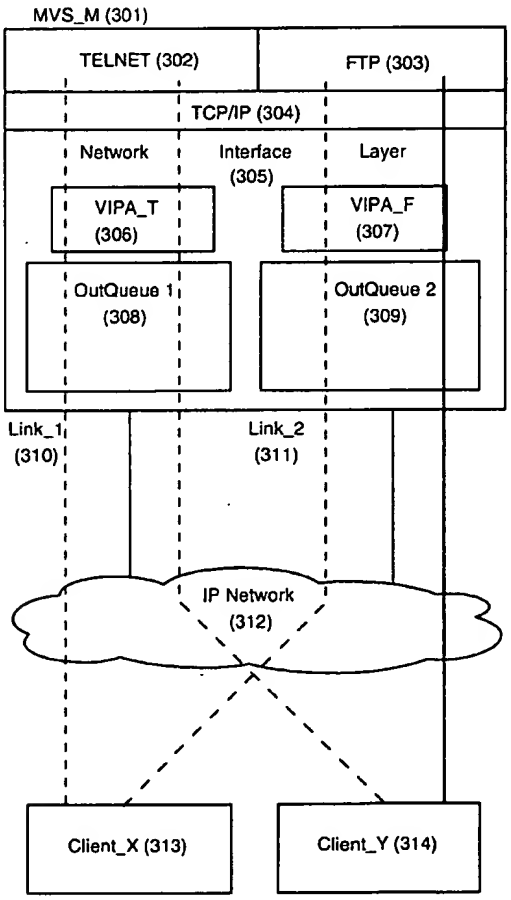
Primary Examiner—Le Hien Luu

Attorney, Agent, or Firm—Jeanine S. Ray-Yarletts

[57] **ABSTRACT**

The present invention utilizes Virtual Internet Protocol Addressing (VIPA) to enable a host computer to efficiently route TCP/IP traffic across multiple physical links. This is accomplished by associating different applications or application sets with different virtual IP addresses. The virtual IP addresses may be associated with different physical adapters. Since many applications send similar data repeatedly, categorizing application sets and associating them with different physical adapters allows high volume applications to be associated with one adapter while lower volume, interactive data is associated with another adapter.

7 Claims, 5 Drawing Sheets



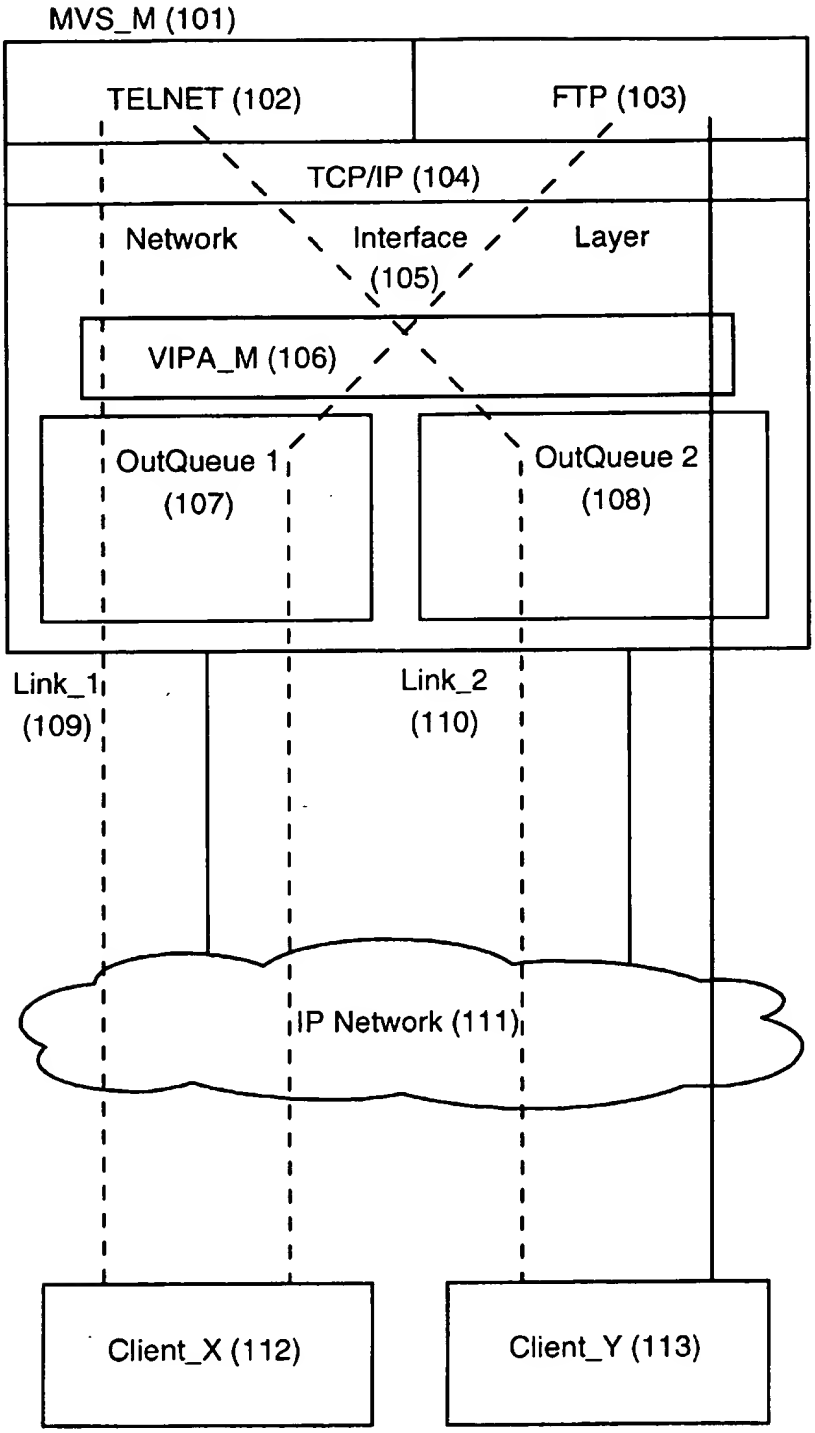

MVS_M (101)
TELNET (102)
FTP (103)
TCP/IP (104)
Network Interface Layer
(105)
VIPA_M (106)
OutQueue 1
(107)
OutQueue 2
(108)
Link_1
(109)
Link_2
(110)
IP Network (111)
Client_X (112)
Client_Y (113)

FIG. 1

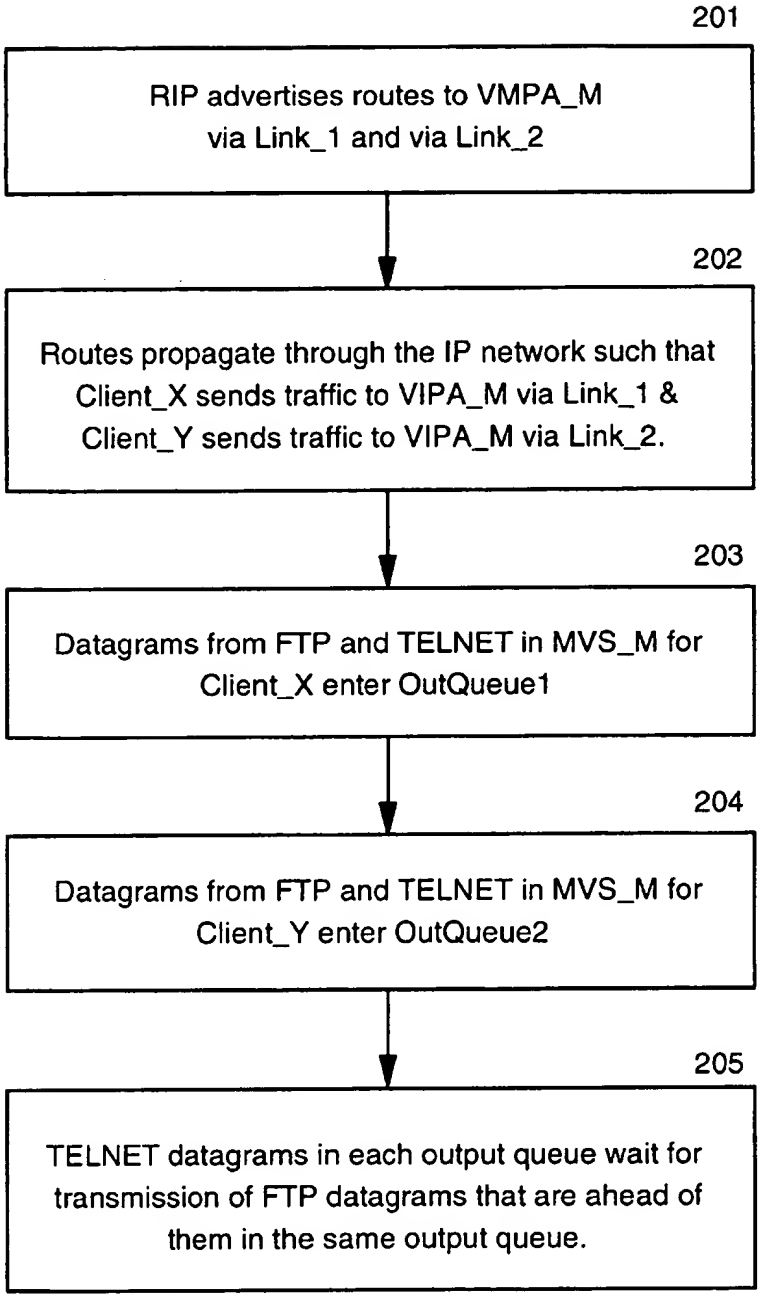

201
RIP advertises routes to VMPA_M via Link_1 and via Link_2
202
Routes propagate through the IP network such that Client_X sends traffic to VIPA_M via Link_1 & Client_Y sends traffic to VIPA_M via Link_2.
203
Datagrams from FTP and TELNET in MVS_M for Client_X enter OutQueue1
204
Datagrams from FTP and TELNET in MVS_M for Client_Y enter OutQueue2
205
TELNET datagrams in each output queue wait for transmission of FTP datagrams that are ahead of them in the same output queue.

FIG. 2

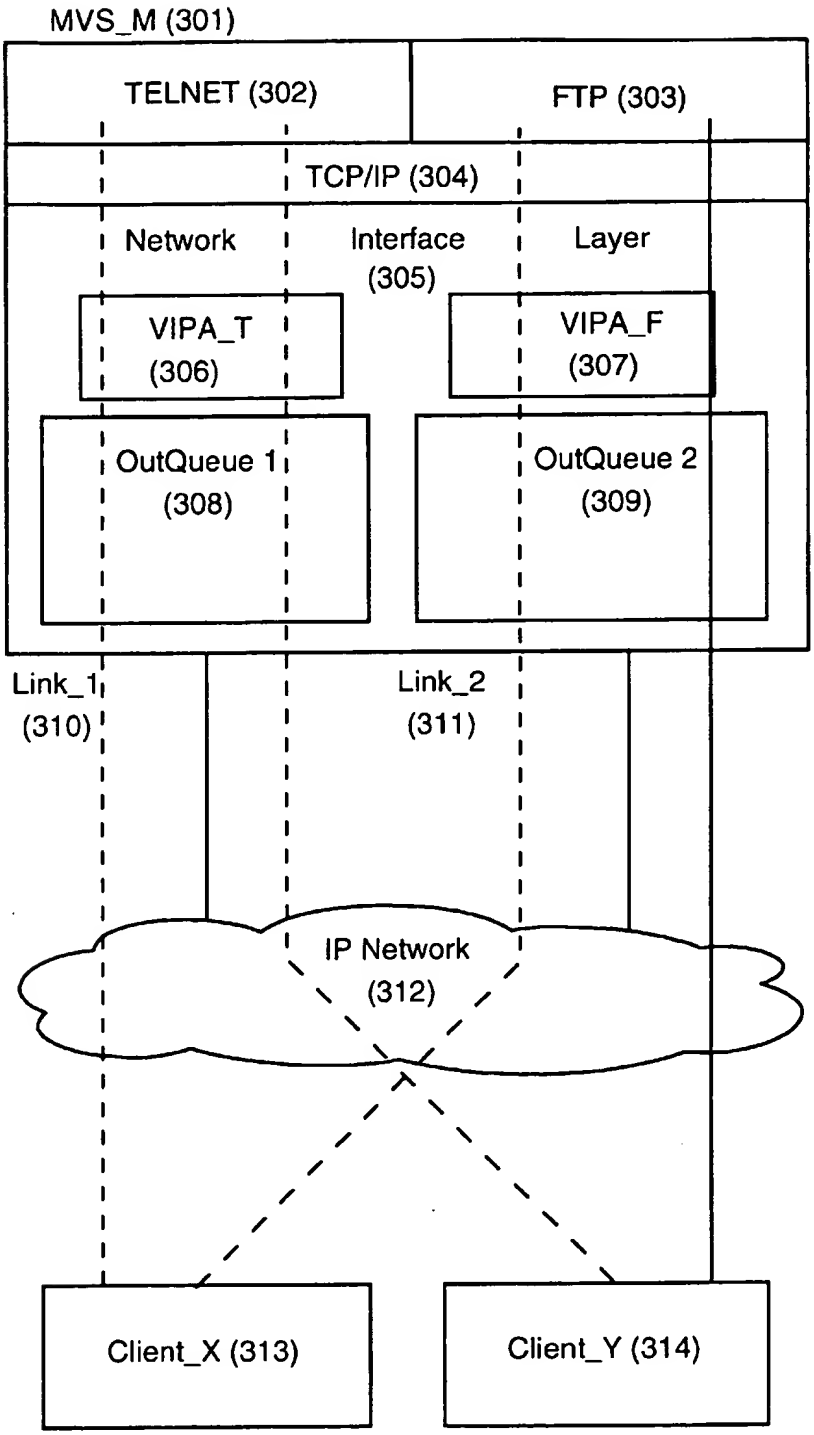

MVS_M (301)
TELNET (302)
FTP (303)
TCP/IP (304)
Network
Interface
(305)
Layer
VIPA_T
(306)
VIPA_F
(307)
OutQueue 1
(308)
OutQueue 2
(309)
Link_1
(310)
Link_2
(311)
IP Network
(312)
Client_X (313)
Client_Y (314)

FIG. 3

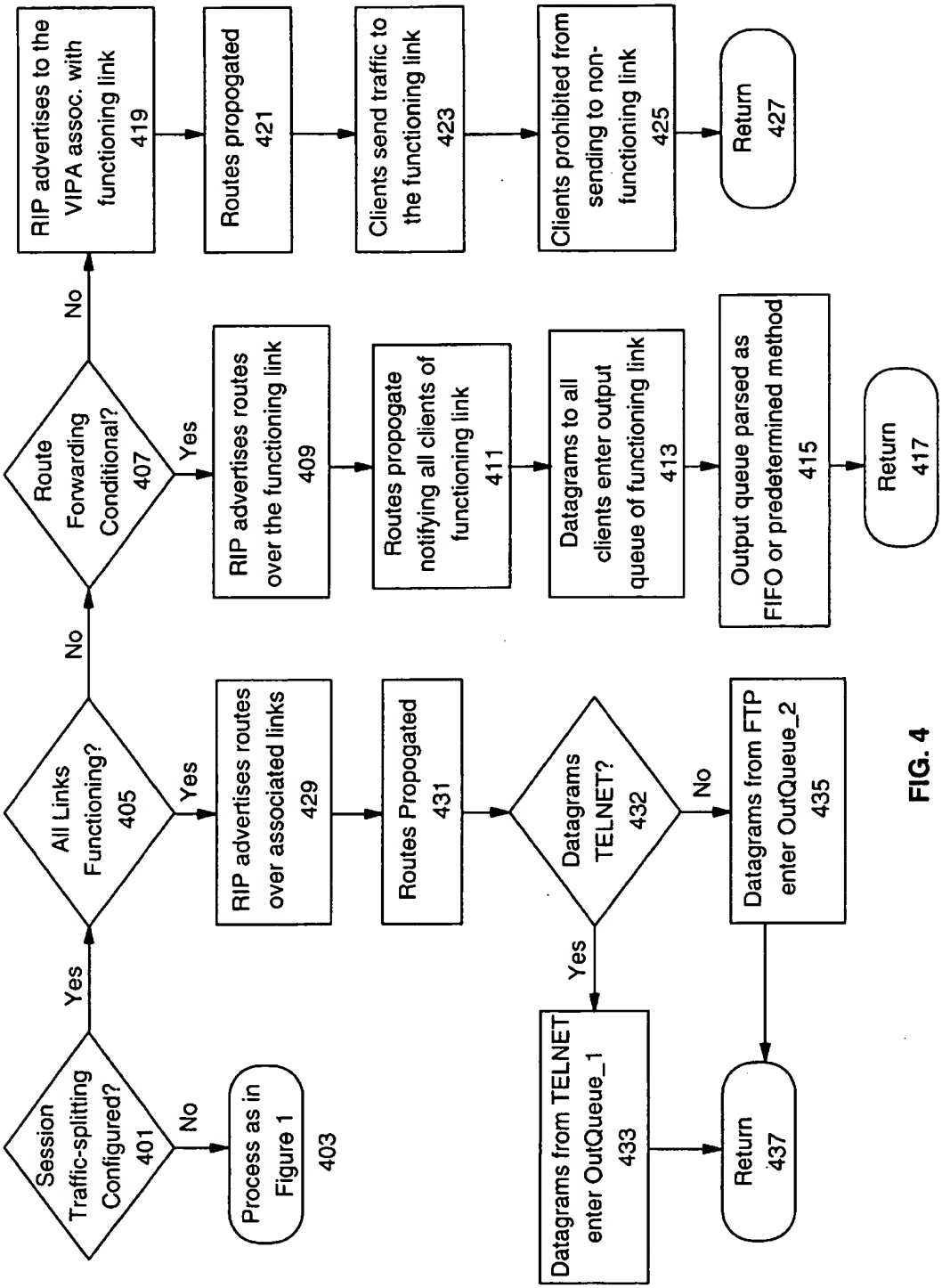

Session Traffic-splitting Configured? 401
Yes
No
Process as in Figure 1 403
All Links Functioning? 405
Yes
No
RIP advertises routes over associated links 429
Routes Propogated 431
Datagrams TELNET? 432
Yes
No
Datagrams from TELNET enter OutQueue_1 433
Datagrams from FTP enter OutQueue_2 435
Return 437
Route Forwarding Conditional? 407
Yes
No
RIP advertises routes over the functioning link 409
Routes propogate notifying all clients of functioning link 411
Datagrams to all clients enter output queue of functioning link 413
Output queue parsed as FIFO or predetermined method 415
Return 417
RIP advertises to the VIPA assoc. with functioning link 419
Routes propogated 421
Clients send traffic to the functioning link 423
Clients prohibited from sending to non-functioning link 425
Return 427

FIG. 4

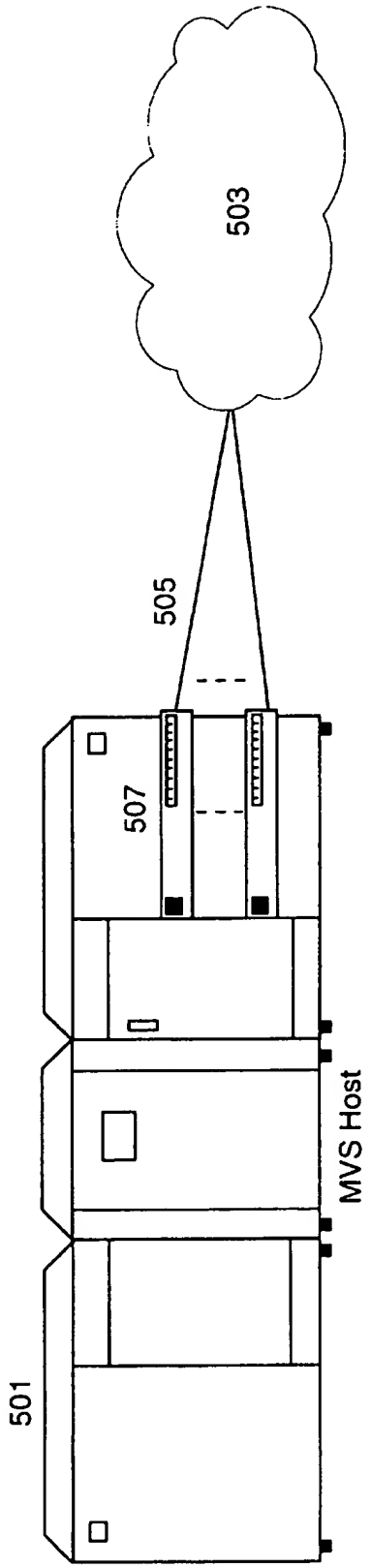


FIG. 5

SESSION TRAFFIC SPLITTING USING VIRTUAL INTERNET PROTOCOL ADDRESSES ASSOCIATED WITH DISTINCT CATEGORIES OF APPLICATION PROGRAMS IRRESPECTIVE OF DESTINATION IP ADDRESS

BACKGROUND OF THE INVENTION

TCP/IP (Transmission Control Protocol/Internet Protocol) is the transport mechanism underlying the Internet. It is also the underlying protocol for many intranets and business applications in existence today. TCP/IP was started as an educational and scientific network. It was not designed to handle high-volume traffic associated with a mixture of applications that have different service level requirements. Because of this design point, there were few effective measures incorporated into TCP/IP to handle applications with conflicting service level requirements.

Through the growth of the Internet, which includes the world wide web, requirements have arisen for better control of service levels for host TCP/IP networks. This has become especially true where the TCP/IP host controls business applications or transactions. The design of TCP/IP is such that each physical network interface adapter has associated with it an address. This is unique within the entire network and is the method by which all other devices communicate with the adapter or the devices connected through the adapter. If a given TCP/IP host has multiple interface adapters, the user communicating with the host must select an interface adapter which they chose to use. The user must then reference the host by the address of the particular adapter which the user has chosen to use. After the routing protocols have converged all the routing tables contained within hosts and routers throughout the network, they all will agree on one route for datagrams from one address on one host to one address on another host.

The above method works well when all applications executing on a given host at one time require the same level of service, or when the interface adapters are not highly utilized, or when the host has only one network interface adapter. However, in many cases a large host system simultaneously executes multiple applications with different service level requirements, has more than one interface adapter available, and highly utilizes some of those interface adapters. For example, when a file transfer protocol (FTP) server is transmitting enough traffic to almost fully utilize some network interface adapters on a host, TELNET (interactive) connections to that server that use those same highly utilized links may suffer from unacceptable interactive response time, because TELNET traffic may have to wait for the transmission across the network interface adapters of the much more voluminous FTP traffic.

Two methods have been proposed for dealing with different service level requirements in a TCP/IP host. First, the Internet Protocol header includes a field with which the host can indicate the type of service requested for the datagram associated with this header. If the Minimum Delay bit in that field is turned on, then all hosts and routers transmitting the datagram should try to transmit it with minimum delay. If the Maximum Throughput bit in that field is turned on, then all hosts and routers should try to transmit it via the highest bandwidth path available. (The header of an IP datagram should not have both the Minimum Delay and the Maximum Throughput bits turned on.) For example, if a host has two links, one a low-bandwidth terrestrial link and the other a high-bandwidth satellite link, the administrator may prefer for high-bandwidth FTP traffic to use the high-bandwidth satellite link despite its inherent delay, and for low-bandwidth but interactive TELNET traffic to use the inherently lower-delay terrestrial link. However, since very few hosts or routers consider the settings of these bits when making routing decisions, turning on either of these bits is not usually an effective procedure. For additional information on the minimum delay bit and the maximum throughput bit see RFC 791, "Internet Protocol".

The second method for addressing the problem is for some hosts have one output queue for datagrams from interactive applications, such as TELNET and rlogin, and another output queue for datagrams from all other applications, including FTP, and have the network interface layer assign datagrams to queues according to the datagram's TCP port number. In some cases, these hosts send out all the datagrams from the interactive queue before sending out any from the non-interactive queue. However, by forcing the high-bandwidth traffic to wait to use the same physical interface adapter as the low-bandwidth traffic, this process results in low throughput if the adapter is slow, or low utilization if the adapter is fast. Network administrators would prefer to have the more voluminous FTP traffic fully utilize a higher-bandwidth (and therefore more expensive) adapter, and have the less voluminous TELNET traffic experience smaller delays over a less highly utilized low-bandwidth adapter, where the lower utilization would matter less because the low-bandwidth adapter is less expensive. Current TCP/IP software limitation do not support traffic splitting, even in the presence of multiple network interface adapters that can reach a common destination. Therefore multiple queues for this purpose, in addition to multiple routes to a common destination are not allowed in the present implementation of TCP/IP. These limitations lead to traffic bottlenecks at the network interface adapters.

RELATED APPLICATIONS

Related applications describing the Virtual IP Addressing (VIPA) technology are:

- Application Ser. No. 08/755,420 entitled "Virtual Internet Protocol (IP) Addressing" filed on Nov. 22, 1996 and assigned to IBM Corp., now pending; and,
- Application Ser. No. 08/761,469 entitled "Host Identity TakeOver Using Virtual Internet Protocol (IP) Addressing" filed on Dec. 6, 1996 also assigned to IBM Corp, now pending.

SUMMARY OF THE INVENTION

The present invention allows the administrator of a host that has multiple network interface adapters to associate each desired subset of applications with a virtual IP address (VIPA) and then configure the host to advertise a route to each such VIPA over a different real physical network interface adapter. This causes traffic for each subset of applications to be transmitted over a different real physical network interface adapter, which the network administrator can select to have bandwidth and delay characteristics that match the application. Therefore, if interactive applications such as TELNET are associated with one VIPA address, and high-bandwidth applications such as FTP are associated with another VIPA, then the voluminous FTP traffic can be segregated to one real physical network interface adapter so that it will not interfere with the interactive response time of the TELNET traffic that is segregated onto a different real physical network interface adapter. Furthermore, this host can be configured such that when a real physical network

interface adapter fails, another one can serve as a backup by integrating the different subsets of traffic for the duration of the link failure.

Without significantly increasing the cost of the network, this invention enhances the performance of a TCP/IP network using hosts with multiple or redundant devices or network interface adapters by eliminating the time that one class of application traffic must wait on another class of application traffic for transmission across network interface adapters, while also providing for backup during the failure of one or more real physical network interface adapters. This is accomplished by the use of a virtual device, a virtual adapter, a virtual IP address (VIPA), and Routing Information Protocol (RIP) route forwarding output filters.

DESCRIPTION OF DRAWINGS

In FIGS. 1 through 4 the host is assumed to have one application subset consisting of TELNET and another application subset consisting of FTP; note that there could be more application subsets, and that any application could belong to any application subset.

FIG. 1 depicts a representative network without session traffic splitting.

FIG. 2 demonstrates the logical flow of the transmission process with prior art.

FIG. 3 depicts the representative network with session traffic splitting.

FIG. 4 demonstrates the logical flow of the transmission process with session traffic splitting.

FIG. 5 shows a representative network in which the present invention may be implemented.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The preferred embodiment of the present invention is implemented in, but not limited to, an IBM MVS (501) host running the TCP/IP protocol and directly connected to a network (503) by two or more links (505) connected to the host through two or more adapters (507). This configuration allows for multiple virtual IP addresses, each of which selects a subset of TCP/IP applications executing on one TCP/IP stack in one host, to proceed without selecting a specific real physical network interface adapter or device. Other hosts that connect to one of the subsets of MVS TCP/IP applications executing on that host can send data to the MVS virtual IP address (VIPA) associated with that subset of applications via whatever paths are selected by the routing protocols. This is enabled by the use of RIP route forwarding filters. Transmission of traffic of one subset of applications between the host and the network will not interfere with transmission of traffic of another subset of applications. Furthermore, should the real physical network interface adapter that is transmitting traffic for a certain subset of applications fail, one of the other real physical network interface adapters can transmit the traffic in a backup mode, although during the outage traffic from different subsets of applications may be mixed. This is explained more completely by way of example in FIGS. 1 and 3.

The means of accomplishing this application categorization and VIPA association is to modify the configuration files of the host's TCP/IP software such that a route to a single VIPA address associated with a desired subset of applications is advertised by RIP (the TCP/IP Routing Information Protocol) only through a real physical network interface adapter specified in the configuration files. To segregate traffic of different subsets of applications, a different VIPA address is associated with each different subset of applications, and RIP is configured to advertise a route to each different VIPA over a different real physical network interface adapter. Furthermore, the host administrator can specify in the configuration files that if a link fails, traffic from the subset of applications that were using that link can share a real physical network interface adapter with traffic from another subset of applications for the duration of the failure. RIP is configured to perform these functions through the Route Forwarding (Conditional and Unconditional) options which provide route forwarding support as a means to filter routes to a particular interface. This can be useful in selecting certain routes to be broadcast over certain interfaces. Two options have been added to support conditional and unconditional route forwarding. With unconditional route forwarding, RIP will not forward routes to other interfaces in case of interface outages. With conditional route forwarding, a forwarded route is allowed to be broadcast over other interfaces throughout the duration of an allowed interface outage. After the failed interface returns to normal, RIP will resume route forwarding to the assigned interface.

FIG. 1 is a representative example of a network that does not use session traffic splitting while still implementing Virtual IP Addressing (VIPA). Host MVS_M (101) has VIPA address VIPA_M (106) by which all clients address host MVS_M. Host MVS_M is executing two applications, TELNET (102) and File Transfer Protocol (FTP) (103), both of which are running on the same TCP/IP stack (104). The TCP/IP stack uses the Network Interface Layer (105) to transmit datagrams to and from the IP Network (111). Within the Network Interface Layer the output queue OutQueue1 (107) holds datagrams ready for transmission across Link_1 (109), and the output queue OutQueue2 (108) holds datagrams ready for transmission across Link_2 (110). Both Link_1 and Link_2 connect to the IP Network. In this ex ample Client_X (112) has connections to both TELNET and FTP via Link_1, and Client_Y (113) has connections to both TELNET and FTP via Link_2. In both OutQueue1 and OutQueue2 Telnet datagrams may be waiting for transmission behind several FTP datagrams.

FIG. 2 demonstrates the process by which the prior art determines routes between MVS_M and the clients Client_X and Client_Y such that each output queue contains datagrams from both TELNET and FTP. In this example, RIP will advertise routes to VIPA_M over both Link_1 and Link_2 (201). In the present example, we assume that (ignoring service levels) the best route between VIPA_M and Client_X traverses Link_1, and that the best route between VIPA_M and Client_Y traverses Link_2. Therefore, after RIP has converged the routing tables throughout the IP network (202), datagrams from VIPA_M to Client_X will pass through Link_1 and therefore through OutQueue1 (203). Likewise, datagrams from VIPA_M to Client_Y will pass through Link_2 and therefore through OutQueue2 (204). Thus, while FTP is sending large files, the relatively small and infrequent TELNET datagrams will be enqueued behind FTP datagrams in both output queues, and therefore transmission of FTP datagrams will impact interactive response times of TELNET users (205).

FIG. 3 is a representative example of a network that uses session traffic splitting of the present invention. Host MVS_M (301) is executing two applications, TELNET

(302) and File Transfer Protocol (303), both of which are running on the same TCP/IP stack (304). The TCP/IP stack uses the Network Interface Layer (305) to transmit datagrams to and from the IP Network (312). Within the Network Interface Layer the output queue OutQueue1 (308) holds datagrams ready for transmission across Link_1 (310), and the output queue OutQueue2 (309) holds datagrams ready for transmission across Link_2 (311). Both Link_1 and Link_2 connect to the IP_Network. Host MVS_M has two VIPA addresses: VIPA_T (306), which is associated with TELNET, and VIPA_F (307), which is associated with FTP. In this example MVS_M has TELNET connections with both Client_X (313) and Client_Y (314) via Link 1, and MVS_M has FTP connections with both Client_X and Client_Y via Link_2. OutQueue1 contains only TELNET datagrams, and OutQueue2 contains only FTP datagrams. The TELNET datagrams, which are destined to be transmitted across Link_1, need not wait for the transmission of FTP datagrams across Link_2.

FIG. 4 demonstrates the process by which this invention segregates all the TELNET datagrams in one output queue such that they are never queued behind FTP datagrams for transmission, which this invention segregates in another output queue. First, this process checks to determine if the administrator has configured session traffic splitting in the host by setting the RIP route forwarding output filter option (401). If not, the processing continues as in FIG. 1 without session traffic splitting (403). If session traffic splitting is configured, then a check is made to determine whether or not all the links are functioning (405). The different types of links are not limited to just terrestrial and satellite, but all imaginable forms of terrestrial and non-terrestrial links. If not, then the process checks whether or not the RIP route forwarding output filter option is conditional (407).

If conditional route forwarding is configured and some links are not functioning, then RIP in MVS_M will advertise over the functioning link routes to both VIPA_F and VIPA_T (409). Therefore, after RIP has converged the routing tables throughout the IP network, datagrams from both FTP and TELNET in MVS_M to any client will pass through the functioning link (411) and therefore through its output queue (413). This results in both the FTP and TELNET datagrams being placed into the same queue. Thus, while FTP is sending large files, the relatively small and infrequent TELNET datagrams will be enqueued behind FTP datagrams in that output queue, and therefore transmission of FTP datagrams will impact interactive response times of TELNET users (415) if the method for parsing the output queue is FIFO, as it is in most TCP/IP implementations. The process is then continued as before (417).

If unconditional route forwarding is configured and some links are not functioning, then RIP in MVS_M will advertise over the functioning link routes to only the VIPA associated with the functioning link (419). Therefore, the routes propagate through the IP network such that, after RIP has converged the routing tables throughout the IP network, all clients send traffic to the VIPA associated with the desired application via the functioning link (421). Datagrams of the application bound to the VIPA associated with the functioning link will pass through the functioning link (423). Furthermore, after RIP has converged the routing tables throughout the IP network, datagrams of the application bound to the VIPA associated with the non-functioning link cannot be delivered (425). From this point, processing continues as before (427).

If the administrator has configured the RIP route forwarding output filter option, and all links are functioning then RIP in MVS_M will advertise routes over their associated links (429). In the present example, MVS_M will advertise routes to VIPA_T only over Link_1 and routes to VIPA_F only over Link_2 (431). Therefore, after RIP has converged the routing tables throughout the IP network all clients will send to the target applications via their associated link. In our particular example, a determination will be made as to whether or not the datagrams are TELNET datagrams (432). If the datagrams are from TELNET in MVS_M they will pass through Link_1 regardless of the associated client and therefore through OutQueue1 (433). Likewise, any datagrams not from TELNET, in our example, are FTP datagrams, whereby datagrams from FTP in MVS_M to any client will pass through Link_2 and therefore through OutQueue2 (435). Thus, even while FTP is sending large files, the relatively small and infrequent TELNET datagrams will not be enqueued behind FTP datagrams in the output queues, and therefore transmission of FTP datagrams will not impact interactive response times of TELNET users. From this point on the process continues as before (437).

What is claimed is:

1. A communications network comprising:

a host computer executing communications applications;

two or more communications interface adapters associated with said host computer;

one communications link attached to each of said two or more communications interface adapters connecting said host computer to a TCP/IP network;

a Virtual Internet Protocol Address (VIPA) associated with each of said communications interface adapters;

an output queue associated with each of said communications links and a corresponding one of the VIPAs; and,

two or more categories of communications applications executing on said host computer such that each VIPA is associated with one or more of said categories of communications applications wherein applications in a category communicate with said TCP/IP network over said communications interface adapter associated with said category of applications utilizing the associated one of the output queues by directing information to said associated VIPA such that communications of different categories utilize different ones of the output queues and corresponding communications links irrespective of a destination Internet Protocol (IP) address of a destination device.

2. A network as claimed in claim 1 wherein:

at least one of said communications links is a terrestrial link and at least one of said communications links is a satellite link.

3. A network as claimed in claim 1 wherein:

at least one category of communications applications is for file transfer and at least one or more categories of communications applications is for interactive traffic.

4. A method of connecting a host computer to a TCP/IP network, said host computer having two or more communications interface adapters, each communications interface adapter attached to one communication link and executing two or more applications, said method comprising the steps of:

defining a virtual Internet protocol (IP) address associated with each of said communications interface adapters;

programmatically categorizing each of said applications programs into distinct categories;

programmatically associating each of said distinct categories of applications programs with a virtual IP address; and,

sending information from any of said applications programs over said TCP/IP network by indicating said associated virtual IP address and utilizing said associated communications interface adapter irrespective of a destination address associated with the information.

5. A method as claimed in claim 4 wherein

at least one of said communications links is a terrestrial link and at least one of said communications links is a satellite link.

6. A method as claimed in claim 4 wherein:

at least one category of communications applications is for file transfer and at least one or more categories of communications applications is for interactive traffic.

7. A programmable media for transferring information to a host computer, said programmable media containing a programmably implemented method of connecting a host computer to a TCP/IP network, said host computer having two or more communications interface adapters, each communications interface adapter attached to one communications link and executing two or more applications, said method comprising the steps of:

defining a virtual internet protocol (IP) address associated with each of said communications interface adapters;

programmatically categorizing each of said applications programs into distinct categories;

programmatically associating each of said distinct categories of applications programs with a virtual IP address; and,

sending information from any of said applications programs over said TCP/IP network by indicating said associated virtual IP address and utilizing said associated communications interface adapter irrespective of a destination address associated with the information.

* * * * *